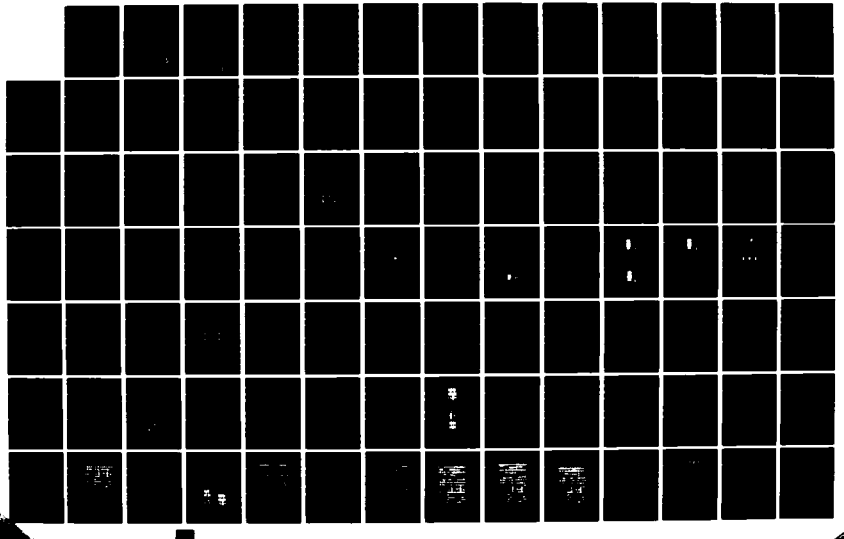
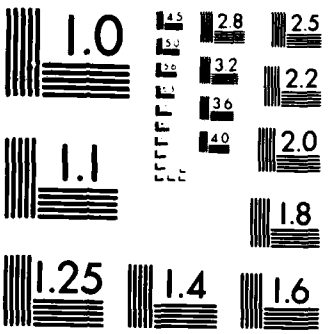
AD-A150 744 IC LAYOUT GENERATION AND COMPACTION USING MATHEMATICAL OPTIMIZATION(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE H WATANABE 1984 TR-128 N00014-78-C-0614 1/2

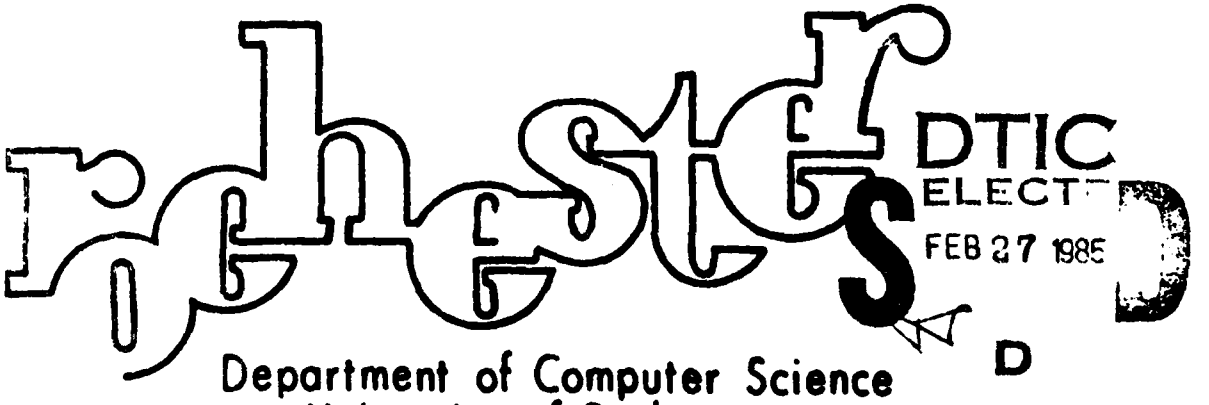
UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2)



Department of Computer Science
University of Rochester
Rochester, New York 14627

IC Layout Generation and Compaction Using Mathematical Optimization

by

Hiroyuki Watanabe

TR 128

Submitted in Partial Fulfillment
of the
Requirements for the Degree
Doctor of Philosophy

Supervised by Gershon Kedem

Department of Computer Science
The University of Rochester
Rochester, N.Y. 14627

1984

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR 128	2. GOVT ACCESSION NO.: AD-A150744
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): IC Layout Generation and Compaction Using Mathematical Optimization	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Hiroyuki Watanabe	8. CONTRACT OR GRANT NUMBER(s): N00014-78-C-0614
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, University of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: 1984 13. NUMBER OF PAGES: 109
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Ph.D. research is in the area of automatic IC mask generation and compaction. It develops a new method of mask compaction. It formulates a mixed integer linear programming problem from a user defined stick diagram, a dimensionless topological representation of IC layout. By solving this mixed integer program, a compacted and design rule violation free layout is obtained. A specialized algorithm was developed for solving this mixed integer program for the purpose of efficiency. This algorithm is based on

FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont.)

a branch and bound method and a longest path algorithm on acyclic graphs.

In this formulation, a vertical graph and a horizontal graph are generated from the input stick diagram. These two graphs are strongly related to each other by binary decision variables. For every fixed set of decision variables, we solve two independent longest path problems, one for the horizontal direction and the other for the vertical one. The interaction between the horizontal and vertical compaction is via the binary decision variables. The branch and bound method is used for setting values of decision variables. The formulation can incorporate the following:

(1) Simultaneous consideration of two directions of compaction
(2) Goal directed compaction and generation of the layout.

The research is aimed at the final end of a spectrum of research for a silicon compiler. The method developed is suitable to be used in the final stage (mask generation stage) of a silicon compiler. For example, having a library of cells in stick diagram form together with a set of procedures to translate them into layout is much more flexible and therefore more useful than a standard cell library. The cells could be scaled automatically according to fan in and fan out considerations. Their shapes could be altered automatically to fit other parts. Moreover, changes in design rules will not render the cells obsolete. The algorithm can also be incorporated into an IC design work station as a compaction procedure.

An experimental program was developed which implemented the compaction algorithm. The program demonstrated that the algorithm is flexible, powerful, and efficient.

Curriculum Vitae

Hiroyuki Watanabe was born April 10, 1953 in Tokyo, Japan. He graduated from Kaisei high school in Tokyo. In April, 1972, he entered the International Christian University (ICU) in Tokyo, where he studied linguistics and languages (English and French). In September, 1974, he came to the United States as an exchange student to the Grand Valley State Colleges in Michigan from ICU. Though the exchange program was for one year, he decided to continue his studies at the Grand Valley State Colleges further. He graduated from there in December, 1975 with a B.S. in mathematics with High Honors. He entered the Department of Computer Science at the Indiana University in August, 1976 receiving a teaching assistantship. He graduated with an M.S. in computer science in May 1978.

He continued his graduate studies in the Department of Computer Science at the University of Rochester. He participated in the first Very Large Scale Integrated Circuit (VLSI) course offered from the University of Rochester in fall 1977. He received his second M.S. in computer science in May, 1980. As a research assistant, he worked with Professer Gershon Kedem in developing, transferring, and organizing the VLSI software. For his doctoral research, Mr. Watanabe developed an algorithm for automatic IC mask layout and compaction. The algorithm uses mathematical optimization techniques. He will continue his research in the design automation of the integrated circuits at AT&T Bell Laboratories.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Gershon Kedem. The technical guidance I received and the long discussions we had in the course of the development of the idea in this thesis were truly valuable.

I would like to express my appreciation to Professor Christopher Brown for reading my draft in detail and giving me helpful advice, and to Professor Greg Dobson for giving me an advice concerning mathematical optimization techniques.

My thanks are also for Marciej Ciesielski and Yoshio Hayashi for useful and exciting discussions we had in the area of design automation and mathematical optimization.

I would also thank Emil Rainero for his comments on recent drafts of this thesis and Rose Peet for correcting my English.

I would like to express special thanks to Frank and Barbera Tiesma for providing me an American home since I started to study in the United States a long time ago.

Finally I am truly thankful to my parents, Hiroyasu and Hisako Watanabe, for supporting me in many ways for eleven years of academic life.

This work was supported in part by the Defense Advanced Research Projects Agency under grand number N00014-78-C-0614 and in part by the Semiconductor Research Corporation under grant number 83-01-028.

Abstract

This Ph.D research is in the area of automatic IC mask generation and compaction. It develops a new method of mask compaction. It formulates a mixed integer linear programming problem from a user defined stick diagram, a dimensionless topological representation of IC layout. By solving this mixed integer program, a compacted and design rule violation free layout is obtained. A specialized algorithm was developed for solving this mixed integer program for the purpose of efficiency. This algorithm is based on a branch and bound method and a longest path algorithm on acyclic graphs.

In this formulation, a vertical graph and a horizontal graph are generated from the input stick diagram. These two graphs are strongly related to each other by binary decision variables. For every fixed set of decision variables, we solve two independent longest path problems, one for the horizontal direction and the other for the vertical one. The interaction between the horizontal and vertical compaction is via the binary decision variables. The branch and bound method is used for setting values of decision variables. The formulation can incorporate the following:

(1) Simultaneous consideration of two directions of compaction

(2) Goal directed compaction and generation of the layout.

The research is aimed at the final end of a spectrum of research for a silicon compiler. The method developed is suitable to be used in the final stage (mask generation stage) of a silicon compiler. For example, having a library of cells in stick diagram form together with a set of procedures to translate them into layout is much more flexible and therefore more useful than a standard cell library. The cells could be scaled automatically according to fan in and fan out considerations. Their shapes could be altered automatically to fit other parts. Moreover changes in design rules will not render the cells obsolete. The algorithm can also be incorporated into an IC design work station as a compaction procedure.

An experimental program was developed which implemented the compaction algorithm. The program demonstrated that the algorithm is flexible, powerful, and efficient.

Table of Content

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1. Introduction

Laying out artwork for the layout design of an integrated circuit is a very tedious task of high economic importance. Consequently, there has been active research in design tools and in design automation of integrated circuits. The research in these areas is being pursued to cope with the complexity of VLSI permitted by new fabrication technologies. The research reported here is an attempt to increase the generality, capability and flexibility of the mask generation process. This research is aimed at the final end of a spectrum of research for a silicon compiler. The method developed is suitable to be used in the final stage (mask generation stage) of a silicon compiler. It can incorporate goals given by a higher level automatic planning process [Hel79,SzO80]. The compaction method can be used directly by humans at an engineering work station. This research introduces a new paradigm in symbolic layout generation algorithms. Given a dimensionless symbolic layout, a mathematical optimization problem is formulated. A compacted and design rule violation free geometric layout is obtained by solving this optimization problem. The mathematical optimization formulation used is mixed integer programming. The formulation is very powerful in the sense that it can incorporate the following:

(1) Simultaneous consideration of two directions of compaction

(2) Goal directed compaction and generation of the layout

(3) Automatic generation of jog points

The first two features are treated uniformly and are considered simultaneously. The best combination of decisions for a given goal is made automatically as a result of the mathematical optimization.

1.2. Symbolic Layout

Symbolic layout is one of the approaches taken for computer aided design. In this approach, the designer is allowed to enter a topological specification of the integrated circuit in a symbolic form. The actual geometric layout is generated automatically. Thus, the designer is relieved from the most tedious and error-prone process of mask level specification. There are two types of symbolic layout methods. They were termed the *fixed grid method* and the *relative grid method* by Min-Yu Hsueh [HsP79,Hsu79]. The former is also called as a *coarse grid method* [GiN76,GiN77].

1.2.1. Fixed Grid Method

In the fixed grid method, the largest of minimum wire width and separation requirement of all layers is treated as a unit length. The layout plane is sliced into unit squares. The designer defines a layer configuration in terms of unit squares (coarse grids). He maps it into a predefined set of characters and enters it into the system. The system automatically transforms

it into a geometrical mask layout. The advantage of this method is that it uses a straightforward algorithm, since there is an one-to-one mapping of the matrix of squares into a geometrical layout. Also, the method does not require a special graphic input device. The disadvantage is that there is a limitation on compactness of the layout produced because of very nature of a coarse grid. The user still performs substantial work because all the geometrical neighboring relationships are still defined by him. Also, the system is incapable of performing any topological or geometric transformation of input and is consequently very inflexible.

1.2.2. Relative Grid Method

The relative grid method was first introduced by Williams. He used a stick diagram for representing an IC layout [Wil77, Wil78]. In a stick diagram, circuit elements such as transistors and butting contacts are represented by symbols. The user is relieved from describing a complex structure of layers which make these circuit elements. In some systems these symbols have actual dimensional size of representing circuit elements [HsP79, Hsu79] and in other systems they are purely symbolic [Dun78, Dun79, Dun80, Mos80, Mos81, Wil77, Wil78]. The interconnections among these circuit elements are represented by dimensionless center lines. According to the layer of the interconnect, they are represented by different colors or different type of lines. The stick diagram is a lower level representation of an electric circuit than traditional schematics. It is much closer to an actual layout. Therefore, the automatic transformation of a stick diagram to a layout is computationally feasible, moreover it is possible to produce a compacted layout comparable to that of a human designer.

The advantage of a stick diagram is that it does not include geometric spacing and dimensional information. Therefore, the designer can specify the circuit layout without considering the tedious spacing requirements of the design rules. He only considers relative topological relations among circuit elements and interconnections. The term *topological* is used for representing relative positions among elements but not their absolute geometrical positions. Also, the relative positions specified by the stick diagram are loose enough so that there is a possibility to perform topological transformations in order to generate a compact layout. However, it is still a non-trivial process and only minimal attempts have been made in the existing systems.

The other advantage of representing a circuit by a stick diagram is related to hierarchical design of integrated circuits. If we can define a library of cells or leaf cells [Row80a, Row80b] in terms of stick diagrams, then we can have flexible and deformable basic cells. The higher level planning algorithms [Hel79, SzO80] or composition cells, which define placement of leaf cells and interconnection among them [Row80a, Row80b], can take advantage of this flexibility. They can produce a goal or a recommendation to the lower level layout generation process which in turn produces a compact layout in a global context.

The design of a VLSI system should be performed hierarchically [MeC80] in order to cope with its complexity. It is possible to design a digital system hierarchically, which increases an efficiency of the design process and produces a cleaner system [McW78a, McW78b]. In the hierarchical method, the design is performed in top-down. However, the actual layout of an integrated circuit is done bottom-up, thereby losing an opportunity to use global information in laying out local cells. Since stick diagrams represent only partial information about the geometric mask layout, we can defer some decision to a later stage. Therefore, stick diagrams can be an aid in producing a layout in a top-down manner.

1.3. Stick Compilers

Although stick diagrams themselves have been very successful in assisting human designers in producing layouts, an automatic conversion of the stick diagrams to the geometric layouts is very attractive and useful. Automatic mask generation can guarantee a layout free from design rule violations and relieve the human designer from the most tedious aspect of integrated circuit design. In order to exploit the concept of deformable cells in the context of the hierarchical design system, an efficient algorithm for generating layouts from stick diagrams is necessary. In addition the algorithm must be able to incorporate goals or recommendations it receives from a higher level composition or a planning process. Useful forms of the recommendation are the aspect ratio for cells and position for connection points.

1.3.1. Existing Systems

After stick diagrams were introduced by Williams [Wil77], several research efforts for an automatic transformation of a stick diagram to a geometrical layout were carried out. The existing systems are: STICKS compiler by Williams [Wil77, Wil78], SLIP and SLIM by Dunlop [Dun78, Dun79, Dun80], CABBAGE by Hsueh [HsP79, Hsu79] and REST by Mosteller [Mos80, Mos81]. All of them perform layout compaction. These systems and their algorithm of layout generation and compaction are described briefly.

1.3.1.1. STICKS Compiler

The STICKS compiler is historically the first of these systems. It performs compaction by dividing a two-dimensional problem to two one-dimensional problems. The algorithm considers only the horizontal or the vertical direction at any one time. The compaction is accomplished by placing the bottom most (or left most) elements on the base line (origin). Then elements on a second horizontal (or vertical) coordinates are placed as close as possible to the elements already placed without violation of design rules. Therefore, the compaction is accomplished by placing elements tightly. The direction of compaction alternates. The spacing and placing algorithm does not look at the global picture. Therefore, the compression in the horizontal direction may block subsequent compression in the vertical direction and vice versa. The system does not create jog points automatically. The designer is required to specify fracture points (a possible place for a jog) in order to insert the jog in the interconnections. The system is heavily dependent on user interaction to accomplish global compaction of the layout. The algorithm itself does not have any provision to incorporate the global goal.

1.3.1.2. SLIP and SLIM

SLIP and SLIM, developed by Dunlop, use compression ridges. Compression ridges are excess area that can be taken out across a chip layout (Figure 1.1). This technique is originally described by Akers [AGR70] for a coarse grid layout. The algorithm considers one direction at a time and the direction of compaction alternates. The symbolic description of layout consists of nodes and lines. The nodes represent circuit elements and the lines represent interconnections. The symbolic layout is transformed to the geometric layout without violating any mask tolerance rules. The transformation is done without any attempt at the compaction. The rows and columns of the symbolic layout remain as rows and columns in the generated geometric layout. Then the compression ridges are formed from the unused area. They should run from one side of the layout to the other side. The compression ridges are not necessarily a single strip but can be the group of strips of excess area connected with shear lines. The shear lines run perpendicular with the compression ridges. At a time of removal of the compression ridge,

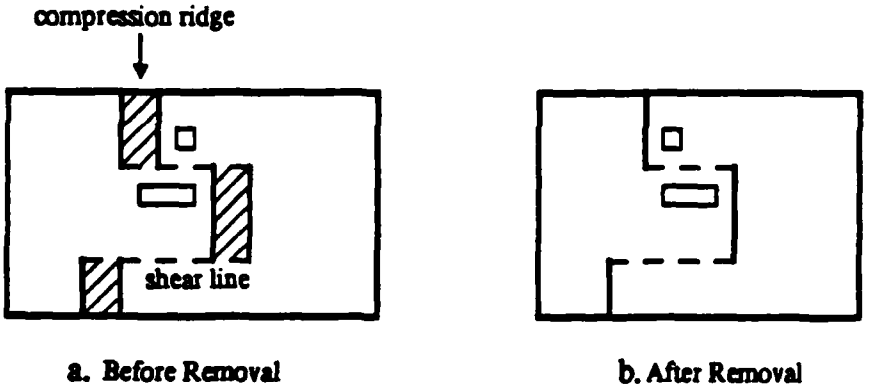


Figure 1.1. Compression Ridge and Shear Line

everything along the shear lines is slid by the width of compression ridge (Figure 1.1). A tolerance testing scheme is used to determine the maximum width of the compression ridge. The formation of the compression ridge is basically a search process of the unused space and a backtracking technique is used. Therefore it is fairly expensive. SLIM is an extended version of SLIP and uses local clustering for improving efficiency. Jog points can be introduced only at nodes and can not be introduced at an arbitrary place of interconnects.

1.3.1.3. CABBAGE

The algorithm used by CABBAGE seems most efficient among these programs. In this system, compaction is done basically by placement. The system concentrates its effort only on the symbolic elements and the interconnects that contribute to the actual layout size. These elements and the interconnects are on the most congested path. The system constructs two graphs called a vertical graph and a horizontal graph. All the elements that share same y-coordinate (x-coordinate) are grouped as one node in the vertical (horizontal) graph. There is an edge between any two nodes directly visible to each other. Two groups, therefore nodes, are visible to each other if there is a straight line segments which intersects both groups without intersecting any other elements [Hsu79,SLM82]. The weight of the edge is the minimum spacing requirement between the corresponding group of elements. The elements that contribute to the actual dimension of the layout are the ones corresponding to the nodes on the longest paths of each graph. The placement of the elements is carried out according to the constraints along the longest path. Again, the horizontal and vertical compacting operations are performed separately and alternately. The jog points are automatically generated along the longest path in order to shorten it.

1.3.1.4. REST

REST has an elaborated input system. A user enters a stick diagram consisting only of line segments. Electrical elements such as transistors and contacts are automatically extracted from the stick inputs. As far as the compaction algorithm is concerned, REST does not provide

an innovative method. The compaction algorithm used is similar to the one used by CABBAGE. It is directed graph based and iterates in the vertical and horizontal directions. REST has a facility to incorporate user defined constraints. Typically, they are related with positioning of connection points to the outside of the layout. The system does handle jog point insertion.

1.3.1.5. MULGA

MULGA is a complete symbolic layout system running under UNIX [Wes81a, Wes81b]. A stick input is mapped into a virtual grid plane. The virtual grid is similar to the coarse grid (fixed grid). In the virtual grid method, the spacing between coarse lines is determined by the interference of neighboring grid locations. The compaction algorithm is not graph based. The compaction is one dimensional and the horizontal and vertical compactions are performed separately. An adjacency information between elements is readily available through the matrix of grid, since each element is associated with grids. This is an advantage of this system. The design rule checking required for compaction can be done locally and therefore rapidly. Graph construction in other graph based algorithms requires substantial portion of computing time. The disadvantage of the grid based system is that it can not compact as tight as the graph based algorithm. The larger the size of symbolic elements compared to the width of wire elements, the larger the wasted space could be.

1.3.2. Problems with the Existing Systems

Here we overview the deficiency of the existing systems and investigate opportunities for further research and development.

All the existing algorithms perform compaction as two one-dimensional problems. The horizontal and vertical compactions alternate. Therefore, there is always the possibility that a compaction in one direction blocks a subsequent compaction in the other direction. More generally, since the algorithms are performed iteratively, an arbitrary sequence of decisions is imposed. Thus, a previous decision restricts the possibility of subsequent decisions. The optimal layout within the restricted domain, defined by a given model, may not be achieved. The model defines the representation of symbolic layout, the translation scheme to the geometric layout, and the possible geometric and topological transformations allowed for compaction, and the criterion for layout efficiency.

Another problem is that the compaction operation is performed only with local information. This is the case for the STICKS compiler and SLIP. In SLIM and especially in CABBAGE, the compaction is guided by a global view which is gained using a longest path of the graph model. However, the compaction operation itself is still basically a local operation. The global guidance is used only to select the place where the operation is applied. Assume there are two or more operations whose combined application improves the layout. If the separate application of any of them can not achieve improvement, then no existing system performs the operations. The global guidance schemes of the existing systems are insufficient even for this moderately sophisticated decision.

There is very weak control over the final layout produced by the compaction operation. None of the operations performed is goal-directed. In goal-directed compaction, the goals given to a system guide the compaction process and the final shape of the cell. Each operation is applied blindly to decrease the overall layout area. However, if a circuit is a sub-block of a larger system, then there is likely be a preferable shape of the layout even though the layout may not be the minimal obtainable. By laying out the sub-block in a certain shape, we can

eliminate a possible dead space between sub-blocks and achieve a smaller layout for the global system. The other goal that has to be considered is the position of connection points. The existing systems simply try to minimize the layout area without considering these factors. Among the existing systems, only CABBAGE has weak control over the direction of compaction.

In order to utilize the layout generator in the context of hierarchical design methods, the system should have a provision to incorporate those goals. The fully hierarchical layout generator not only incorporates the goals given but also arranges the generation process in a hierarchical manner. The system should adjust the goals for each sub-block from the information of their placement and the connection among them. Other important information is the current possibilities of each sub-block to be conformed with the current goals. Therefore, the local goals for each sub-block should be dynamically adjusted in the process of compaction in order to achieve the global goal at the top level of the hierarchy.

1.4. Research Objectives

This section outlines our research objectives, which are generally to overcome the deficiencies of the existing compaction systems described in the previous section.

The first and most important objective is to accomplish true two dimensional compaction. Both directions of compaction, horizontal and vertical directions, should be considered simultaneously. Also, an arbitrary order of decisions should not be imposed in a compaction operation. The compaction operation, therefore, should not be iterative.

The compaction should be guided by global information rather than local information. Operations should be deployed which together can improve the overall layout, although they may not accomplish any local improvement.

Goal directed compaction is another purpose of this research. The compaction method should be able to incorporate objectives or goals given by a user or a higher level process. The goals that should be incorporated are concerned with the shapes of the cell and the positions of the connection points. It should be done elegantly with minimum overhead. These goals give a user a strong control over the compaction process and the final layout. In case goals are not satisfiable, the method should try to comply with them as much as possible. Then, it should report the failure to satisfy the goals.

We should not treat these objectives individually with ad hoc methods. Instead, to accomplish these objectives uniformly, we need to formulate a general algorithm to solve the compaction problem.

CHAPTER 2

Mixed Integer Linear Programming

2.1. Mathematical Optimization for Layout Generation

In this chapter, methods of linear programming and mixed integer programming are explored as possible techniques for layout generation and compaction.

The formulation described here is an attempt to overcome the deficiencies discussed in the previous chapter. The formulation is general enough so that all deficiencies are treated uniformly in a framework of mixed integer programming. There have been attempts at automatic layout design using mathematical programming in a slightly different context [OSK70, Otv75, ZiS74]. Branch and Bound [OSK70], mixed integer programming [ZiS74] and linear programming [Otv75] are used for determining the shape of sub-blocks or islands (isolated regions) under the assumption that their topological placement is given. Also, the use of the mixed integer programming in the symbolic layout of the integrated circuit is suggested by Hachtel [Hac81]. A graph theoretic algorithm is used in a model for digital circuit optimization by Leiserson, Rose and Saxe [LRS83].

Before starting, we will discuss the assumptions concerning the representation of the layout and the symbolic library. The layouts consist of wires and circuit elements such as depletion mode transistors, enhancement mode transistors and butting contacts. The circuit elements are defined in the library and they are connected by wires in the layout.

(1) The layout is represented in the first (positive) quadrant of the Cartesian coordinate system.

(2) The symbolic representation of a circuit element is the minimum enclosing rectangle of the layout of the element.

(3) The placement of instance i of a circuit element in the layout is represented by the coordinate of its central point (x_i, y_i).

(4) w_i, h_i are the half-width and half-height of an instance i of a circuit element.

(5) H_{ij}, V_{ij} are the horizontal and vertical spacing requirements between symbolic instances i and j of circuit elements.

2.2. Linear Programming

Design rules are defined in terms of a minimum spacing requirement between two circuit elements, a minimum width of interconnection wires, and a minimum overlap of different layers. Therefore it is possible to represent the design rules as a set of inequalities. Ignoring the interconnections, for simplicity, the requirement of a legal layout for the two neighboring symbols in Figure 2.1 is represented by the following inequality:

$$(x_2 - w_2) - (x_1 + w_1) \geq H_{12}.$$

Moving constants to the right hand side,

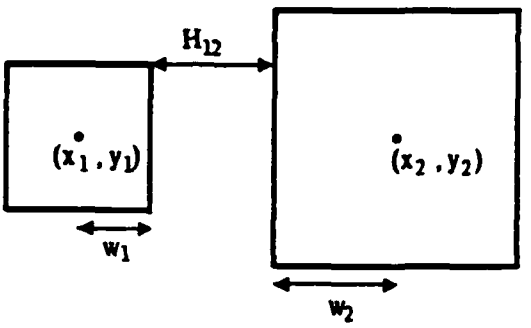


Figure 2.1. Constraints between Symbols

$$x_2 - x_1 \geq H_{12} + w_1 + w_2.$$

By representing all of the constants by d_{12},

$$x_2 - x_1 \geq d_{12}. \tag{2.1}$$

Similar inequalities can be formulated for all circuit elements that might be visible to each other during the process of compaction. If all of the inequalities are satisfied, the layout will be free from design rule violations. The objective function to be optimized under these constraints is the area of the resulting layout, $A = W \cdot H$. H is the coordinate of the highest segment and W is the coordinate of the right most segment in the layout. The area A is not linear but quadratic. We can use the approximation, $W+H$, as suggested by Otten [Otv75]. Then, we can formulate the linear programming problem:

minimize $W+H$

$\mathbf{Ax} \geq \mathbf{b}, \ \mathbf{x} \geq \mathbf{0}$

where **A** is a coefficient matrix obtained from the left hand side of inequalities such as (2.1), and **x** is a vector whose value represents the layout, and **b** is the vector obtained from the right hand side of inequalities such as (2.1). This linear programming problem appears to generate the optimum layout under these constraints. However, linear programming is not general enough to handle the following situations related to the two-dimensionality of the compaction operation.

Case 1:

In the situation depicted in Figure 2.2, the symbol B can slide either in the horizontal or vertical direction relative to the symbol A. Depending on the surrounding conditions, one of the movements may contribute more for the overall compaction of the layout than the other. The design rule requirements for this case are represented as the following:

$x_b - x_a \geq H_{ab} + w_a + w_b$ horizontal constraints

or

$y_a - y_b \geq V_{ab} + w_a + w_b$ vertical constraints .

The requirement is that at least one of the above two conditions should be met. However, this is not linear. The legal region for the central point for B, (x_b, y_b), is not convex as shown in

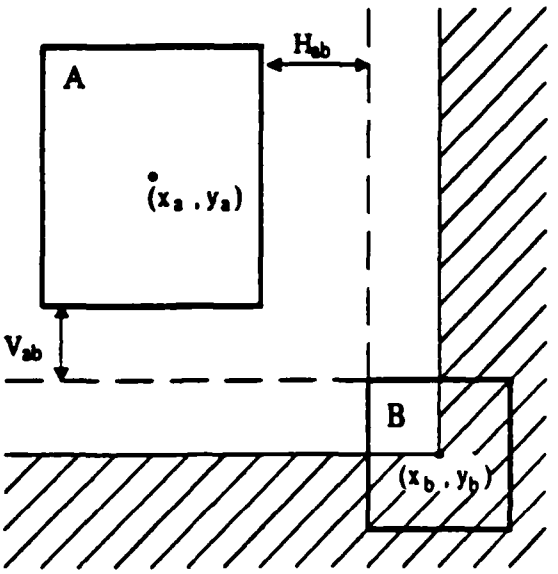


Figure 2.2. Non-Linearity of Compaction Operation

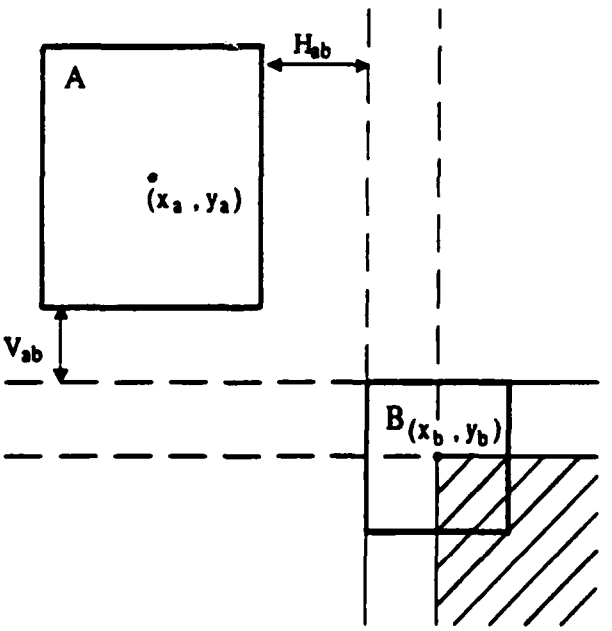


Figure 2.3. Linearized but Over Constrained

Figure 2.2 as the shaded area. If the above conditions are treated as regular linear constraints, the legal region for the symbol B is as shown in Figur 2.3. It is over constrained and is inadequate for the compaction. At best, we have to choose the direction of movement (sliding) of the symbol B at the formulation time of the inequalities. That is, only one of the two conditions is generated. In Figure 2.4, only the vertical constraint is generated and the symbol B is able to move in the horizontal direction only.

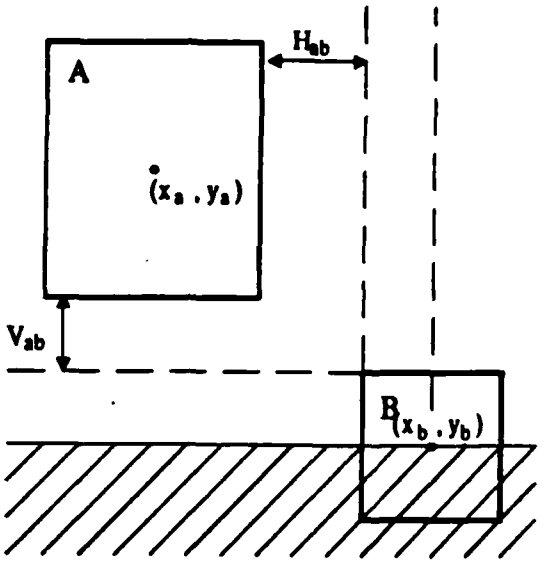


Figure 2.4. Only Horizontal Direction Is Chosen

The linearity is preserved but the compaction operation is still severly restricted by the arbitrary decision made at the problem formulation time.

Case 2:

The second problem arises when the horizontal and vertical directions have to be considered simultaneously. The situation is depicted in Figure 2.5. The horizontal constraint for symbol C is different according to the relative vertical position of symbol C to symbols A and B. Only one of the constraints from symbol A or symbol B is active at a time. The condition is expressed as follows:

if $y_c - y_b \geq V_{bc} + h_b + h_c$

then $x_c - x_a \geq H_{ac} + w_a + w_c$

else $x_c - x_b \geq H_{bc} + w_b + w_c$.

The above set of inequalities is not linear and the legal region for the central point of symbol C is not convex. It is shown as the shaded area in Figure 2.5. If these two horizontal constraints are taken simultaneously regardless of the vertical relation, the condition becomes linear. The situation is shown in Figure 2.6. It is not adequate for compaction. If there are two or more constraints from same direction, the severest one is always taken as the constraint for all positions, regardless of the actual design rule requirement. A similar situation arises between

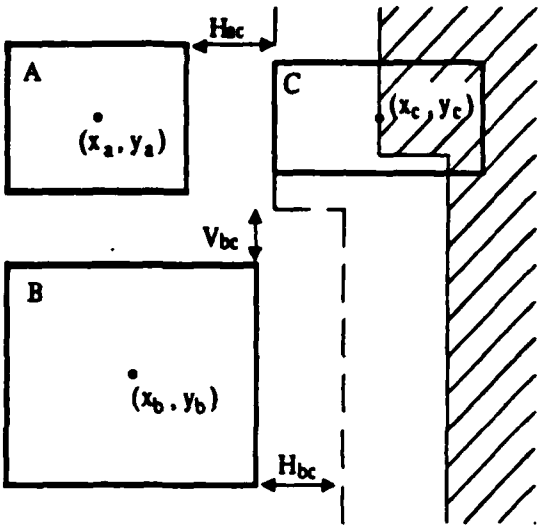


Figure 2.5. Conditional Constraints

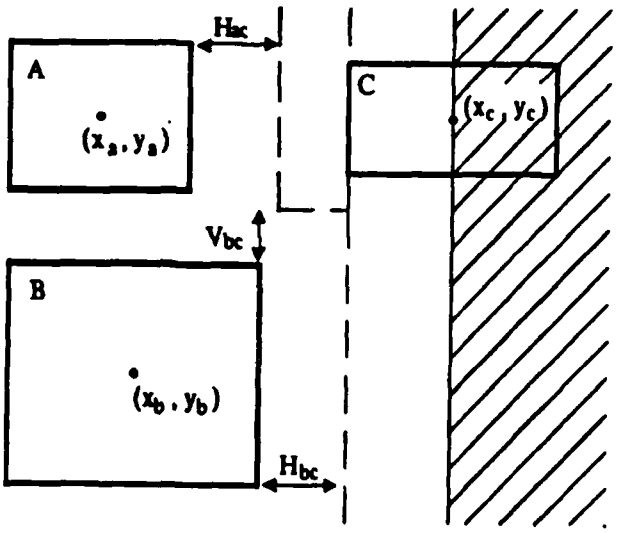


Figure 2.6. Linearized Conditional Constraints

two symbols when the horizontal space requirement between them varies according to their relative vertical positioning, as shown in Figure 2.7.

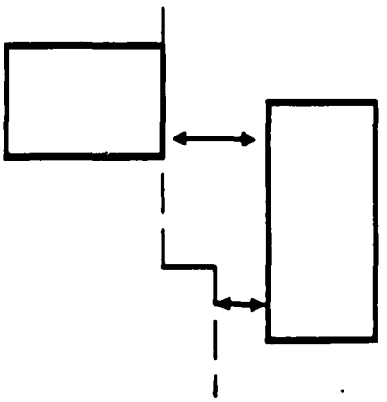


Figure 2.7. Varying Horizontal Space Requirement

In order to use linear programming for layout generation and compaction, we have to make a certain decision about the direction of compaction at problem formulation time. This causes a loss of generality in the compaction operation. A dead area can not be avoided in certain cases such as the second case above.

2.3. Mixed Integer Programming

In order to cope with the above problems, we can use the technique of mixed integer programming [GaN72, p10]. A formulation presented here gives an underlying model for our compaction method. Consider the problem

minimize $f(\mathbf{x})$,

$\mathbf{x} \in S = \{\mathbf{x} | A\mathbf{x} \geqslant \mathbf{b}, \mathbf{x} \geqslant \mathbf{0}\}$

and

$g(\mathbf{x}) \geqslant 0$ or $h(\mathbf{x}) \geqslant 0$ or both .

This problem is not linear because of the dichotomy. However, the dichotomy is equivalent to

$g(\mathbf{x}) \geqslant \delta K_g$

$h(\mathbf{x}) \geqslant (1-\delta) K_h$

$\delta = 0, 1$

where K_g, K_h are known finite lower bounds on g and h, and δ is the 0, 1 integer variable. Using this method, the first case of the previous section can be formulated as the following:

$$x_b - x_a - (H_{ab} + w_a + w_b) \geq \delta K_h$$

$$y_a - y_b - (V_{ab} + w_a + w_b) \geq (1-\delta) K_v$$

$$\delta = 0, 1$$

where K_h, K_v are lower bounds of the left hand side of the above expressions. In practice, we can use a sufficiently small number for the lower bounds.

The second case of the section 4.1 can be formulated using the same technique, because the conditional expression

$$\text{if } g(\mathbf{x}) \geq 0 \text{ then } h_1(\mathbf{x}) \geq 0 \text{ else } h_2(\mathbf{x}) \geq 0$$

is equivalent to the expression

$$g(\mathbf{x}) \geq 0, \; h_1(\mathbf{x}) \geq 0$$

or

$$g(\mathbf{x}) < 0, \; h_2(\mathbf{x}) \geq 0 .$$

This expression is similar to the previous case and equivalent to

$$g(\mathbf{x}) \geq \delta K_g, \; h_1(\mathbf{x}) \geq \delta K_{h_1}$$

$$g(\mathbf{x}) < (1-\delta) C_g, \; h_2(\mathbf{x}) \geq (1-\delta) K_{h_2}$$

$$\delta = 0, 1$$

where K_g, K_{h_1}, K_{h_2} are known finite lower bounds of g, h_1, h_2 and C_g is a known upper bound of g. For computational purpose, the expression with equality $g(\mathbf{x}) \leqslant (1-\delta) C_g$ can be used rather than $g(\mathbf{x}) < (1-\delta) C_g$ without any effect on the result. The conditional expression imposed by the design rule in the second case is formulated as:

$$y_c - y_b - (V_{bc} + h_b + h_c) \geq \delta K$$

$$x_c - x_a - (H_{ac} + w_a + w_c) \geq \delta K$$

$$y_c - y_b - (V_{bc} + h_b + h_c) < (1-\delta) C$$

$$x_c - x_b - (H_{bc} + w_b + w_c) \geq (1-\delta) K$$

$$\delta = 0, 1$$

where K is a sufficiently small number such that all the lower bounds of the above three expressions are greater than or equal to K, and C is a sufficiently large number. The advantage of this formulation is that the compaction is now performed as a single two-dimensional problem rather than two separate one-dimensional problems. Also, all of the local constraints are considered simultaneously for achieving the global optimum. Therefore, all of the local movements are performed in the context of the global goal. The disadvantage is that mixed integer programming is much harder to solve than linear programming. It is known to be NP-complete. There is no known efficient algorithm for integer programming comparable to the simplex method for linear programming. Furthermore, each introduction of a 0-1 variable doubles the search space. So integer variables are fairly expensive.

2.4. Further Generality of Mixed Integer Programming

Mixed integer programming can be used to formulate the selection of the most suitable layout from alternative layouts of a circuit element. One can also use it to formulate rotational choices of circuit elements as part of the compaction operation. Using mixed integer

programming, it is possible to make only one of a set of constraints hold. Consider the following problem:

$$\textit{minimize } f(\mathbf{x}), \quad \mathbf{x} \in S = \{\mathbf{x} | A\mathbf{x} \geq \mathbf{b}, \ \mathbf{x} \geq 0\}$$

and one of the constraints

$$g_i(\mathbf{x}) \geq 0, \qquad i = 1, \ldots, m$$

must hold. The above condition can be replaced by the constraint set

$$g_i(\mathbf{x}) \geq \delta_i K_{g_i}, \qquad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \delta_i = m - 1$$

$$\delta_i = 0, 1, \quad i = 1, \ldots, m$$

where K_{g_i} is a known lower bound of g_i.

If there are m alternative layouts for a single circuit element, then we can generate m sets of constraints. Each set corresponds to one layout of the circuit element. Using the above technique, we can make only one set of constraints active. The best layout is chosen by solving the formulated mixed integer programming. For the rotation of a circuit element, two or more sets of constraints can be generated. Each corresponds to the rotational orientation of a single circuit layout.

2.5. Difficulty with Mixed Integer Linear Programming

The model based on mixed integer linear programming is very general as described in the previous sections. The problem with this formulation is that solving a general form of mixed integer linear program is very costly. We expect a fairly large number of 0-1 variables, possibly in the hundreds or even thousands.

Mixed integer programming problems can be solved by a separation method (Partitioning). The method is called the Benders's decomposition algorithm [Ben62]. The decomposition algorithm separates the mixed integer programming problem into an integer programming part and a linear programming part. The method was successfully used in a distribution problem [GeG71]. The implicit enumeration method [Bal65,Geo69] was used to solve the integer part of the problem. However, the decomposition method is very sensitive to problem formulation, and slight changes in formulation can effect its efficiency [GeG71].

A more common class of methods for solving mixed integer programming problems is the branch and bound method. One branch and bound algorithm uses a linear relaxation method at each step of the branching operation [LaP73]. In this method the integer condition is relaxed. At each node of the branch and bound search tree, the method solves the linear programming problem where the integer variables, δ's, are allowed to take on real values between 0 and 1. According to the solution of the relaxed linear programming problem, one integer variable, δ_i, is chosen as a branching variable. Each branch after separation represents smaller mixed integer programming problems where δ_i is eliminated from the problem; one branch corresponds to the problem where δ_i is set to 1, and the other corresponds to the problem where δ_i is set to 0. Leaf nodes correspond to problems where all δ variables are fixed. The simplex method can be used to solve the linear programming problem. Though the simplex method is empirically known to be fairly fast algorithm, it is still an expensive operation if used repeatedly. It is important to reduce the required amount of computation at each stage of the branch and bound method.

Because we have a large problem that has a special structure, we did not pursue these general methods of solution. Rather, we imposed some restrictions on the problem formulation and constructed a special purpose algorithm based on heuristics and branch and bound methods. In the next chapter, we will develop a refined model which sacrifices some generality to achieve faster computation.

16 - Blank

CHAPTER 3

Graph Optimization

3.1. Introduction

In this chapter, we formulate the branch and bound method using graph optimization concepts. It is a direct refinement of the formulation given in the previous chapter. The new formulation uses the branch and bound method for deciding values of integer variables, and it uses the longest path algorithm instead of linear programming at each stage of the branch and bound method. It also takes advantage of the special structure of our problem. A similar graph theoretic model and the longest path algorithm is used for digital circuit optimization by Leiserson, Rose and Saxe [LRS83] in context of retiming.

3.2. Input Convention

Let us quickly review the assumptions about symbolic input. The input consists of symbols and their interconnections. These symbols are enhancement transistors, pull-up transistors, and poly-metal contacts, etc.. The symbols are represented by their minimum enclosing rectangles of their layouts. A single interconnecting wire (or line) must be a straight line without any bend. A wire must be in either the horizontal or vertical direction, and diagonal wires are not allowed. A wire segment must connect to other elements at both ends. These terminating elements can be symbols, wires in other directions, or the surrounding borders of the layout. The length of a wire segment is decided by the coordinates of these terminating elements. It behaves as if it were a rubber band.

3.3. Graph Nodes

Two related graphs are constructed whose nodes represent the x and y coordinates of symbols and connecting wires of the layout. The correspondence between the input layout and the nodes of the generated graphs is as follows. For each symbol, we assign two variables; one for the x-coordinate and the other for the y-coordinate. These variables correspond to the nodes of the graphs on which the compaction algorithm operates. A connecting wire between symbols is assigned only one variable. If a wire is vertical then it has an x-coordinate variable and if it is horizontal then it has a y-coordinate variable. The relation is depicted in Figure 3.1. Two separate but interrelated graphs are constructed. All x-coordinate variables have corresponding nodes in a horizontal graph, and y-coordinate variables in a vertical graph. Our compaction algorithm is based on the inter-relation of the two graphs.

Even though these two graphs are directed graphs (digraphs), they are designated by the term *graph* in this thesis. There are four special nodes which correspond to the four borders of the whole layout. Left and right borders have x-coordinates, therefore they have corresponding nodes in a horizontal graph. They represent the source and the sink of the graph. Top and bottom borders have corresponding nodes in a vertical graph and they represent the source and the sink of that graph.

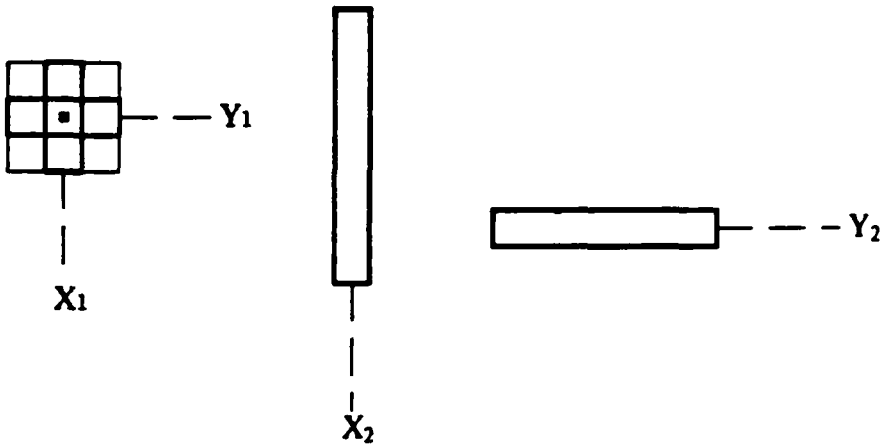


Figure 3.1. Coordinate Variables

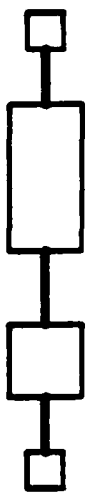


Figure 3.2. Single Vertical Group

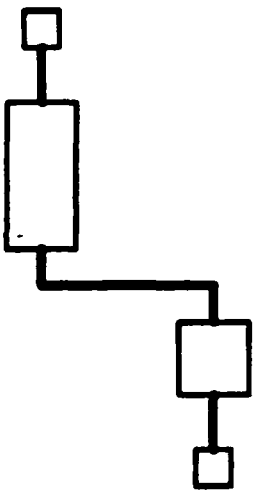


Figure 3.3. Two Vertical Groups

3.4. Arcs in the Graph

Arcs correspond to the linear inequalities formulated in the previous chapter. They are also used for representing connection requirements.

There are two kinds of arcs in the graphs. One of them corresponds to minimum distance requirements between elements in a layout. They are called separation arcs. The other kind of arcs represents connections between symbols and wires and they are called connection arcs. The former type of arcs are further sub-divided into two classes; simple arcs and dual arcs.

We introduce a definition of groups in the following section. The groups are important for deciding directions of arcs.

3.4.1. Vertical and Horizontal Groups

A group consists of symbols and their connecting wires in one direction. For example, a vertical group consists of symbols connected only by a series of vertical wires. Elements in Figure 3.2 represent a single vertical group, while elements in Figure 3.3 represent two vertical groups and one horizontal group. The horizontal wire in Figure 3.3 constitutes one horizontal group by itself. It also separates two vertical groups. Note that vertical groups appear in the horizontal graph and horizontal groups appear in the vertical graph.

A total order is assinged to vertical groups and horizontal groups. This ordering is used to make the graphs acyclic. A coordinate for each group is calculated by averaging central coordinates of member elements; use x-coordinates for a vertical group, y-coordinates for a horizontal group. The groups are ordered by their coordinates. If two different groups have the same average coordinate, one is arbitrarily chosen as a predecessor of the other. The resulting group ordering mostly reflects a topological relation among groups in a given input. The horizontal and vertical groups for the shift cell example are shown in Figure 3.4.

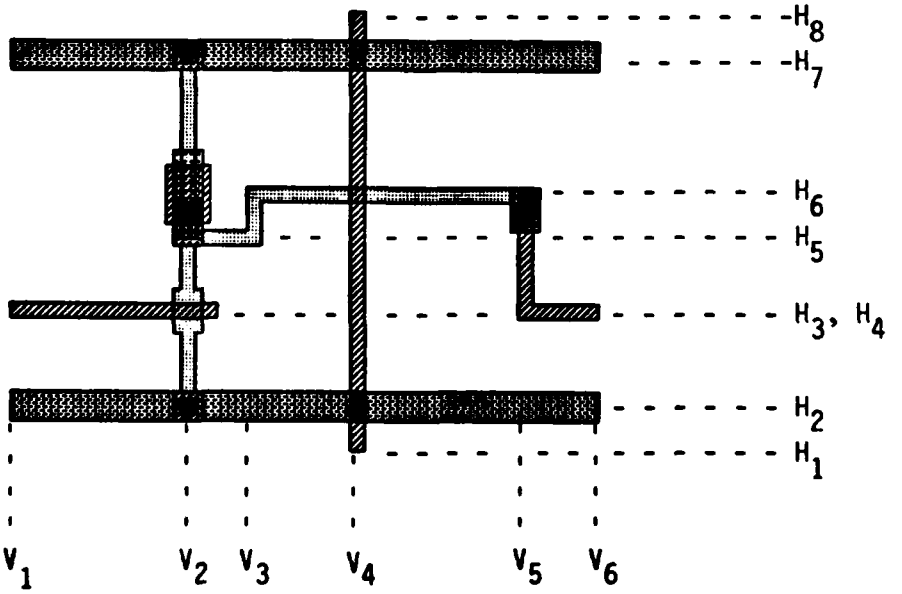


Figure 3.4. Vertical and Horizontal Groups for Shift Cell

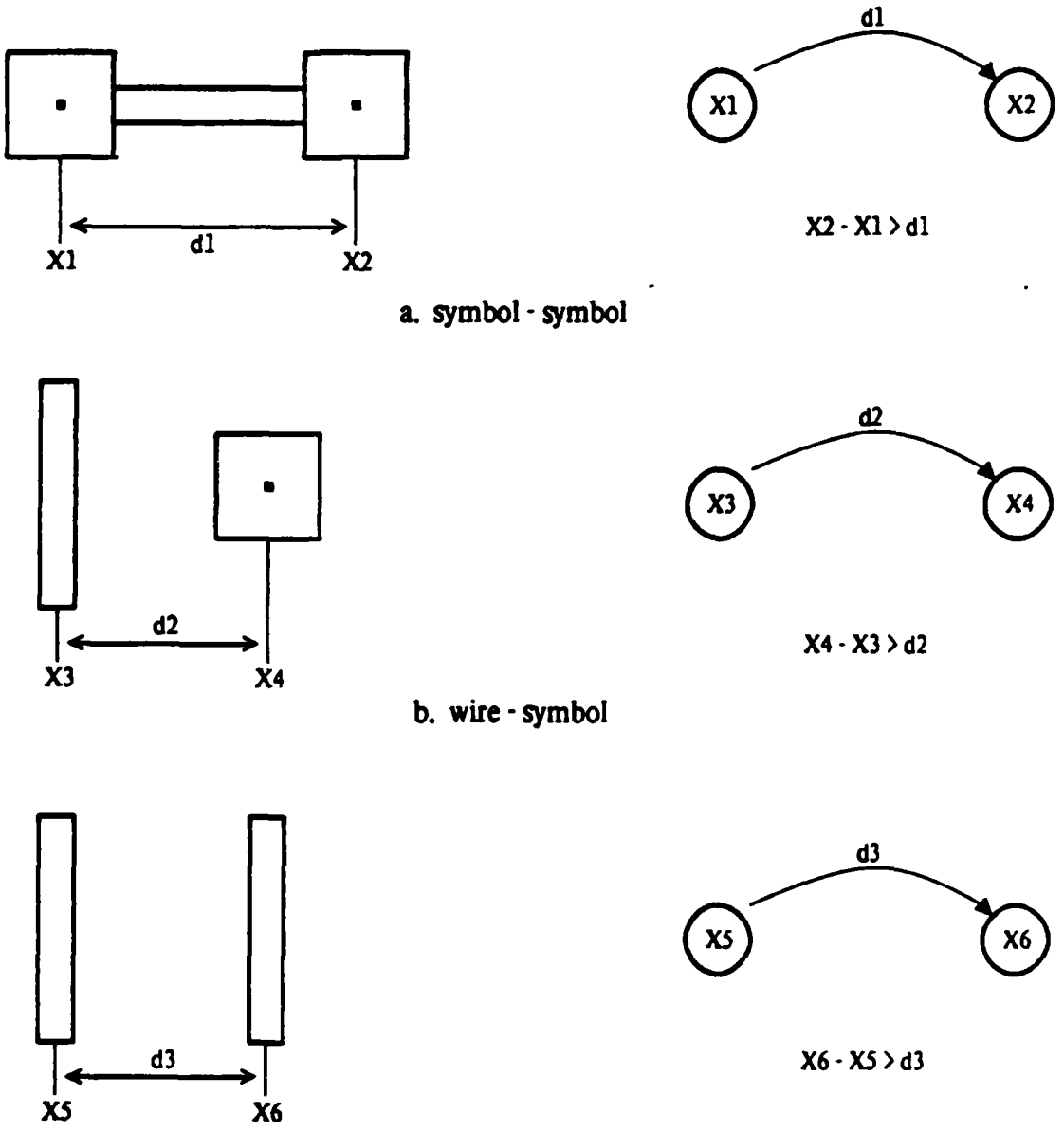


Figure 3.5. Simple Arcs

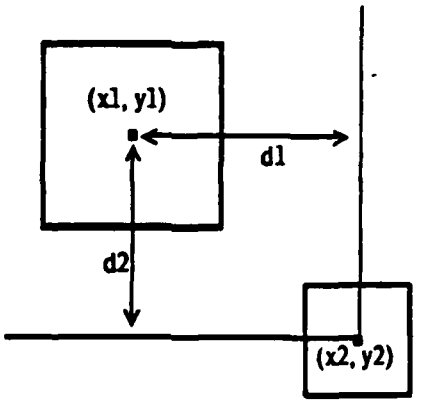
3.4.2. Distance Requirements Imposed by Design Rules

A separation arc in the graph corresponds to the minimum distance requirement between two elements. The head node and the tail node of the arc represent the two elements involved, and the distance is represented as the weight of the arc. Minimum distance requirements are defined so that an arc points from a node in a predecessor group to a node in a successor group

according to the group ordering. Arcs usually point from left to right and bottom to top.

We make the graphs enforce the design rules. If all requirements represented by arcs have been met, then the resulting layout is free of design rule violations. The graph generation algorithm has to create an arc between any pair of symbols for which there is a chance that they can violate a design rule by being too close.

If two symbols are directly connected by a single straight wire, the distance requirement is only in one direction as shown in Figure 3.5a. This is an example of a simple arc generated



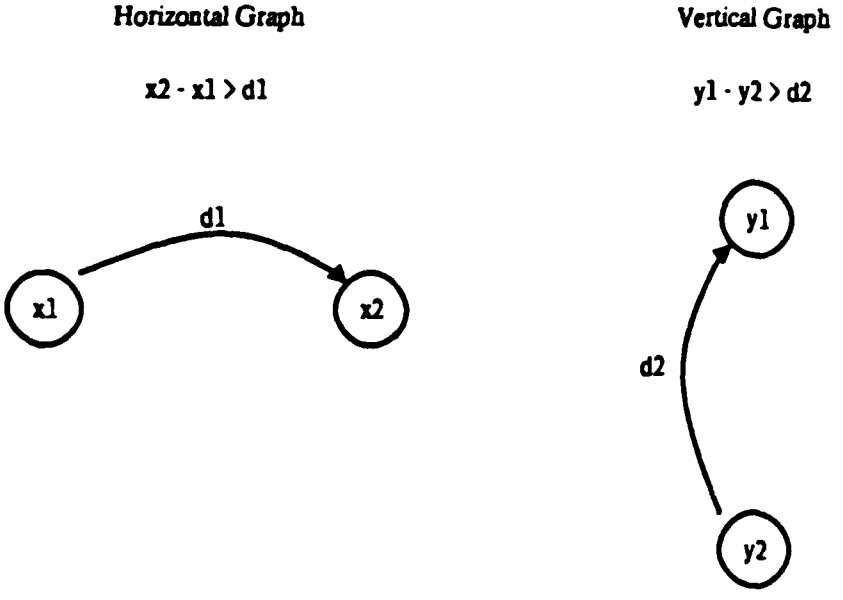
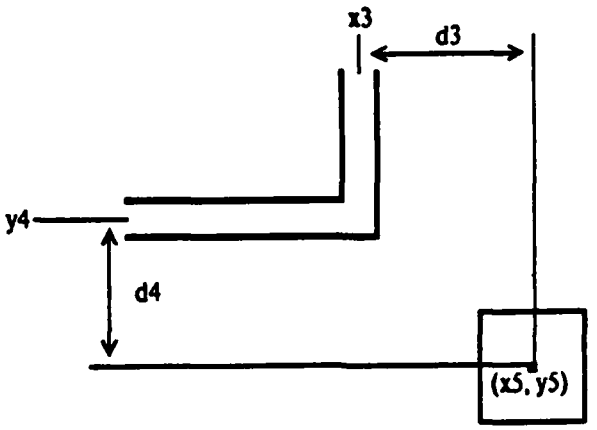


Figure 3.6a. Dual Arcs: Symbol-Symbol

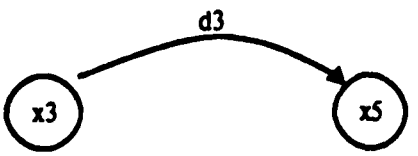
from design rules. Simple arcs are also generated between a wire and a symbol (Figure 3.5b) and between wires (Figure 3.5c).

When two symbols are not connected by a single straight wire but are located close to each other, there is a pair of constraints as shown in Figure 3.6a. One of them is in the vertical direction and the other in the horizontal direction. A pair of arcs must be constructed; one in a vertical graph and another in a horizontal graph. These pairs of arcs are called dual arcs. For



Horizontal Graph

$x5 - x3 > d3$



Vertical Graph

$y4 - y5 > d4$

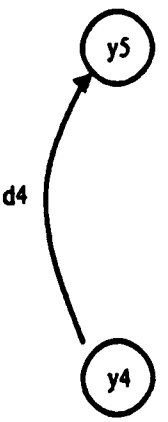
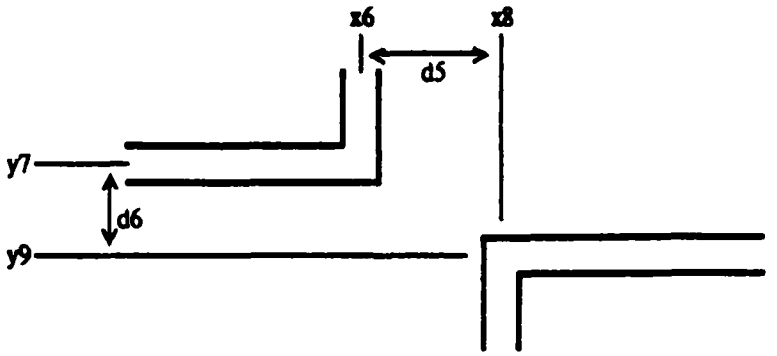


Figure 3.6b. Dual Arcs: Symbol-Wire

the resulting compacted layout to be free of design rule violations, only one of two constraints need to be satisfied. This is because of the distance matrix used (i.e. Manhattan distance) and because all symbols are represented by minimum surrounding rectangles. Similar constraints corresponding to dual arcs are generated between a wire segment and a symbol and between two wire segments, as shown in Figure 3.6b and 3.6c. Since a wire segment has a coordinate variable for only one direction, a coordinate variable of its terminating element must be used for the other direction.



Horizontal Graph

$x8 - x6 > d5$

Vertical Graph

$y7 - y9 > d6$

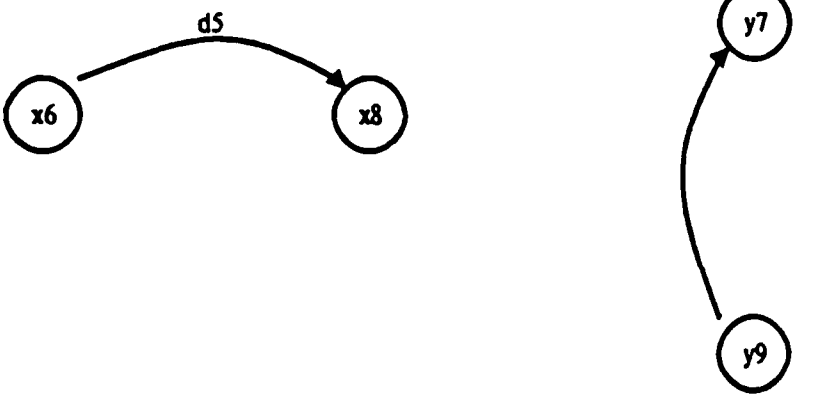


Figure 3.6c. Dual Arcs: Wire-Wire

The important point is that both the vertical and horizontal graphs consisting of separation arcs (i.e. simple and dual arcs) are acyclic. This is because the direction of arcs is decided according to the ordering of the groups. The tail node of an arc always belongs to a group preceding its head node in the ordering.

3.4.3. Connection Constraints between Symbols and Wires

In the course of compaction, it is necessary to maintain connections between symbols and their connecting wires and among wires that are connected. A pair of constraints is used to represent a single connection requirement. This method also allows a wire and its connected symbol to shift relative to each other if the symbol's connection node is wide enough.

We explain the representation of the connection requirement between wires and symbols using an example shown in Figure 3.7.

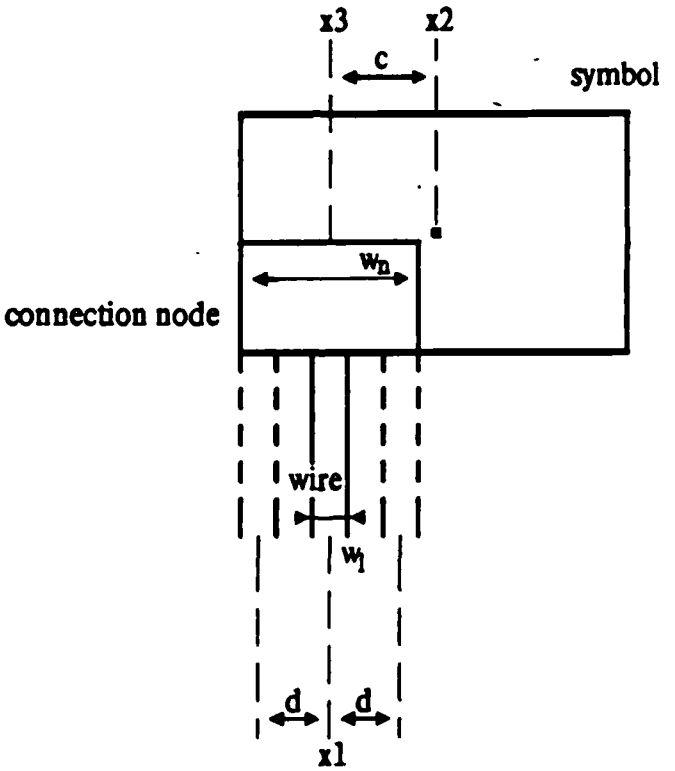


Figure 3.7. Connection Constraints

Let x_1, x_2, x_3 be the center coordinate of a wire, a symbol and a connection node of the symbol respectively, and let w_n, w_l be the width of the connection node of the symbol and the wire. Now define $d = (w_n - w_l)/2$. Then the connection requirement is $|x_1 - x_3| \leq d$, which is equivalent to

$$x_1 - x_3 \leqslant d$$

$$x_1 - x_3 \geqslant -d .$$

Because of the convention that the weight of arc represents minimum distance, we rewrite the above as:

$$x_3 - x_1 \geqslant -d$$

$$x_1 - x_3 \geqslant -d .$$

If the center of the connecting node x_3 coincides with the center of the symbol x_2, then the above pair of inequalities is the required condition. This simpler case is shown in Figure 3.8.

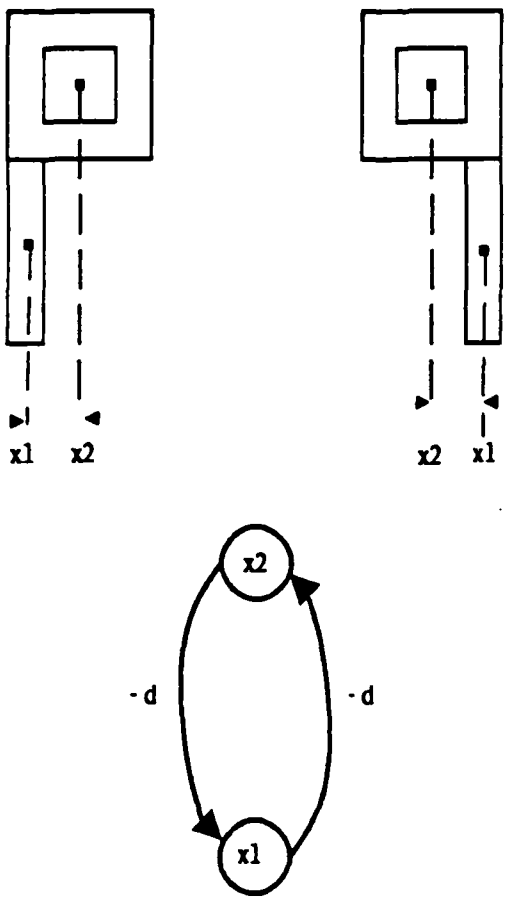


Figure 3.8. Simple Connection Constraints

If it is not the case, then $x_3 = x_2 - c$ for some constant c and we have

$$x_2 - x_1 \geq -d + c$$

$$x_1 - x_2 \geq -d - c \ .$$

The resulting arcs form a cycle between nodes which represent the wire and the symbol (Figure 3.9).

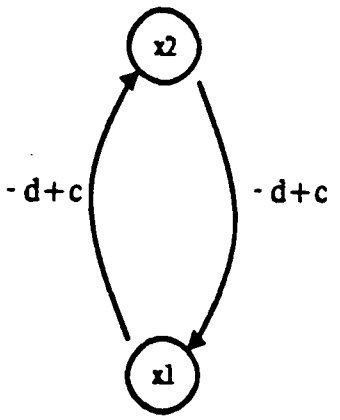


Figure 3.9. Connection Arcs in Horizontal Graph

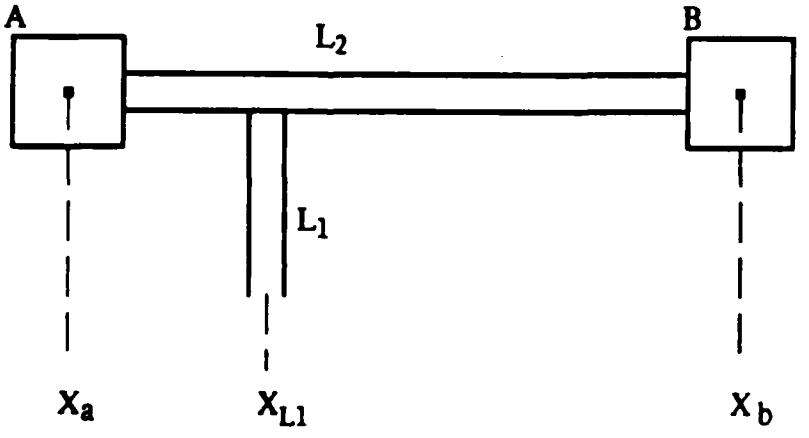


Figure 3.10. Connection between Wires

The connection requirement among wires is slightly different. Figure 3.10 shows a vertical wire L_1 connected to a horizontal wire L_2. The horizontal wire is connected to symbols A and B. The central coordinate of the vertical wire L_1 has to be between the terminating elements of line L_2 to maintain the connection. Therefore, the connection requirements between L_1 and L_2 are represented by the following two inequalities:

$$x_{l_1} - x_a \geq d_a$$

$$x_b - x_{l_1} \geq d_b$$

where d_a and d_b are distance requirement between symbol A and wire L_1 and between symbol B and wire L_1 respectively. They not only secure a connection between two lines but also separate the vertical wire and two terminating symbols. Therefore, these arcs are treated as separation arcs rather than connection arcs. No special arc is necessary at a connection between a wire and a border.

Since the connection arcs destroy the acyclic character of the graphs, they need special handling in the longest path part of our compaction algorithm.

3.4.4. Arcs and Groups

The arcs representing a connection requirement of a vertical wire with its terminating elements are present in the horizontal graph because they constrain a horizontal movement of connected elements. A layout and its horizontal graph are shown in Figure 3.11 and in Figure 3.12. The vertical groups appear as the alternating nodes and arcs, and they form a chain of pairwise cycles in the horizontal graph. The horizontal groups appear in the vertical graph. Note that connection arcs connect nodes in the same group, whereas separation arcs, which represent the design rule requirement, connect nodes in different groups. Within the same group it is only necessary to generate arcs among closely neighboring elements. For example, consider a vertical group which appears in a horizontal graph and not in a vertical graph. For elements within the same vertical group, it is necessary to generate only the following: simple arcs among neighboring symbols in the vertical graph (they represent the minimum distance requirement among them) and connection arcs in the horizontal graph between connecting symbols and wires. It is not necessary to create arcs between other members of the same group. They are redundant.

3.5. Formalization as a Graph Optimization Problem

In the previous sections, we explained the format of the input and construction of the graphs. We now formally introduce the graph based optimization problem. We introduce a 0-1 decision vector δ related to the dual arcs in section 3.5.2..

3.5.1. Notation for Representation of Graphs

Two graphs, a horizontal and a vertical graph, are denoted by G_x and G_y. G_x is defined as

$$G_x = \{N_x, E_x, W_x\}$$

where N_x, E_x, W_x are sets of nodes, arcs, and weights respectively.

$$N_x = \{x_i\}$$

is a set of nodes in the graph. Two special nodes, corresponding to the left and right borders, are represented by s_x (source) and t_x (sink).

$$E_x = \{<x_i, x_j> \mid x_i, x_j \in N_x\}$$

is a set of arcs in the graph. E_x consists of three sub-sets:

$$E_x = A_x \cup D_x \cup R_x .$$

A_x is the set of simple arcs. They correspond to simple horizontal constraints. D_x is the set of

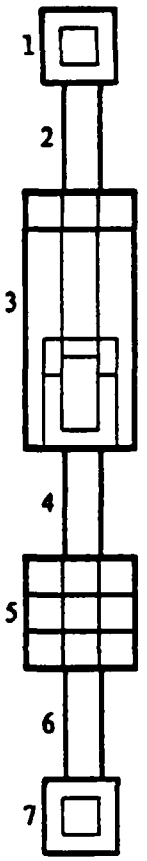


Figure 3.11. Single Vertical Group

Figure 3.12. Vertical Group in Horizontal Graph

horizontal parts of dual arcs. For each element of D_x, there is a corresponding vertical arc as a dual pair. R_x is a set of connection arcs for the vertical groups.

$$W_x = \{w_{ij}^x \mid <x_i, x_j> \in E_x, w_{ij}^x \in \mathbf{R}\}$$

is a set of weights. Here **R** is the set of real numbers. A weight is attached to each arc, and it represents either a horizontal distance requirement or an amount of freedom of movement at a connection point.

The vertical graph G_y is defined similarly:

$$G_y = \{N_y, E_y, W_y\}$$

where N_y, $E_y = A_y \bigcup D_y \bigcup R_y$, W_y are defined analogously for the vertical direction.

3.5.2. Decision Variables

Let I be an index set into D_x and D_y. If $i \in I$ then $e_i^x \in D_x$, $e_i^y \in D_y$, and $<e_i^x, e_i^y>$ is a pair of dual arcs. By construction, D_x and D_y have the same number of arcs.

For i-th pair of dual arcs, let us introduce decision variables δ_i. Each decision variable can take a value from {0, 1}. We denote a vector of decision variable by δ.

3.5.3. The Optimization Problem

We now formulate the optimization problem. For each node $x_i \in G_x$ and $y_i \in G_y$, we assign a value x_i, y_i, representing positions of symbols and wires in a layout. We use the node name and its value interchangably in this section.

The vector $\mathbf{z}=<\mathbf{x}, \mathbf{y}, \delta>$ is said to be feasible if the following three conditions are satisfied:

(1) For each arc $e \in A_x \bigcup R_x$; $e=<x_i, x_j>$

$$x_j - x_i \geqslant w_{ij}^x.$$

(2) Similarly for each arc $e \in A_y \bigcup R_y$; $e=<y_i, y_j>$

$$y_j - y_i \geqslant w_{ij}^y.$$

(3) For each $i \in I$, let i-th pair of dual arcs be $<e_i^x, e_i^y>$ where $e_i^x=<x_j, x_k>$ and $e_i^y=<y_j, y_k>$. Then

$$x_k - x_j \geqslant w_{jk}^x \quad \text{if } \delta_i=0$$

$$y_k - y_j \geqslant w_{jk}^y \quad \text{if } \delta_i=1.$$

Let $\Omega=\{\mathbf{z} \mid \mathbf{z}=<\mathbf{x},\mathbf{y},\delta>;\ \mathbf{z} \text{ is feasible}\}$ be the set of feasible points. Our optimization problem is:

$$\min_{\mathbf{z} \in \Omega} f(\mathbf{x},\mathbf{y}) \tag{3.1}$$

where $f(\mathbf{x}, \mathbf{y})$ is the objective function. Ordinarily the objective function is the area occupied by the layout. That is $f(\mathbf{x},\mathbf{y})=(t_x - s_x)(t_y - s_y)$. Without loss of generality, we will assume $s_x=s_y=0$, therefore, $f(\mathbf{x},\mathbf{y})=t_x \cdot t_y$.

The graphs G_x and G_y encode systems of inequalities. They are related to each other by the dual arc pair $<e_i^x, e_i^y>$ and corresponding decision variables δ_i for each $i \in I$. The compaction algorithm of the layout is to find the 0-1 vector δ with which vectors x and y take values minimizing $f(\mathbf{x},\mathbf{y})$. We use a branch and bound method to decide the values of δ and a longest path algorithm to find the values of x and y. The method for compaction is described in Chapter 5.

3.6. Restriction of The Formulation

The ordering of the groups and generation of separation arcs according to this ordering is a restriction we imposed on our method. Because of this restriction, there is no positive cycle in the graphs G_x and G_y. A positive cycle is a cycle of arcs whose weights sum up to a positive number. Furthermore, their subgraph, G'_x and G'_y, without connection arcs, are acyclic graphs. G'_x and G'_y are defined as

$$G'_x = \{N_x, A_x \bigcup D_x, W_x\}$$

$$G'_y = \{N_y, A_y \bigcup D_y, W_y\} .$$

The only cycles in the graphs G_x and G_y are pairwise loops consisting of arcs in the set R_x and R_y, and for all of them, the sums of weights of arcs in each loop is non-positive. Therefore it is possible to use a method which is a slight modification of a longest path algorithm on acyclic graphs.

What we sacrificed to gain this advantage is the ability to formulate the three way decision or four way decision of relative positioning of two symbols. We can only have one choice in each direction, horizontal and vertical. The example in Figure 3.13 represents two symbols A and B.

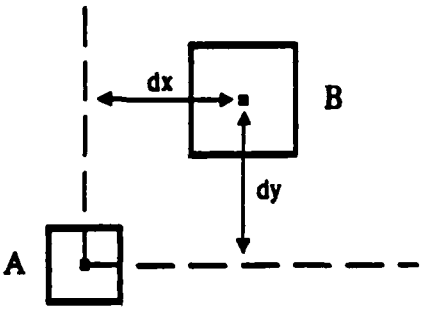


Figure 3.13. Two Symbols A and B

Their possible relative positions are also indicated. The dual constraints generated are as follows:

$$x_b - x_a \geq d_x$$

$$y_b - y_a \geq d_y .$$

With these constraints, symbol A is not allowed to move up to the right side of symbol B. Therefore, the positioning shown in Figure 3.14 never occurs.

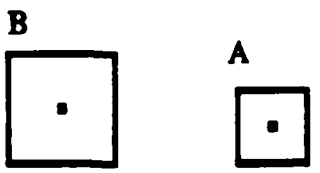


Figure 3.14. Impossible Position for Symbols A and B

Now assume two symbols are connected to wires as shown in Figure 3.15. Obviously, the above dual constraints are less than desirable. The better constraints, which are likely to give two symbols more freedom to move relative to each other, are the following:

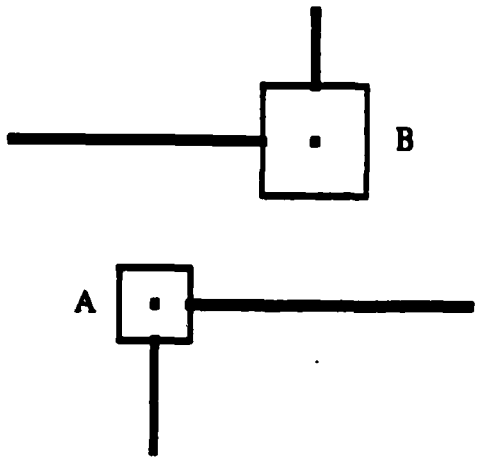


Figure 3.15. Two Symbols and Their Connected Wires

$$x_a - x_b \geq d_x$$

$$y_b - y_a \geq d_y .$$

The direction of the horizontal arc is reversed. However, the first constraints can not simply replaced by the second. In order to change the direction of the horizontal arc between symbols A and B, it is necessary to reverse the direction of all arcs between the two groups to which symbols A and B belong. Otherwise, a positive cycle may be generated. It is possible to design an algorithm to reorder the groups considering not only relative positions but also connections among symbols and wires. This overcomes the problem with this particular example. However, the basic restriction, the fact that we can formulate only two way choices, remains. A possible remedy using an iterative approach is presented in Chapter 6.

CHAPTER 4

Graph Generation

4.1. Introduction

In this chapter, problems and algorithms related to generating a graph from a stick diagram are considered. The design rules are those specified by Mead and Conway [MeC80] and supplemented by Lyon [Lyo80]. The design consists of the following six layers: polysilicon, diffusion, metal, cut, implantation, buried cut. A loosely drawn CIF layout is used for the stick diagram. The stipple pattern we use for drawing is shown in Figure 4.1

Figure 4.1. Stipple Pattern for NMOS Layers

4.2. Electrical Node Extraction

In order to generate a graph, we need to extract the electrical nodes from the stick diagram. We need this information to calculate a minimum distance requirement between elements. For example, there is no separation requirement between two different components of the layout if they belong to the same layer and to the same electrical node.

Since our stick diagram consists of symbolic elements and connecting wires, we need to perform mask level electrical node extraction only once for each symbolic element. If library symbols are predefined, information for electrical nodes can be extracted in advance and can be stored in the library.

The symbolic elements are represented by constituent rectangles of each layer. We use a subset of CIF [HoS80, MeC80] for this purpose. Only the rectangles (Box command in CIF) are allowed in our representation. The Wire and Polygon commands are not used. Different rectangles are given the same electrical node number if they are electrically connected. Some rectangles in the diffusion layer may be reduced or separated into two rectangles, if they are overlapped by polysilicon rectangles. We also create rectangles for channel area; channel rectangles are areas where rectangles in diffusion and polysilicon layers are overlapped. Note that channel area coincides with gate area. We use a fictitious layer "channel" for representing channel rectangles. Rectangles in the channel layer are not considered to be electrical nodes, and no electrical node numbers are assigned. For the node extraction operation, we use a table-driven rectangle based algorithm [Ter80].

Figure 4.2 shows an example of an enhancement mode transistor whose length/width ratio is 1/2. The data structure that represents this symbolic element is shown in Figure 4.3. It consists of two rectangles; one polysilicon rectangle and one diffusion rectangle. These rectangles directly correspond to the Box commands of CIF. By applying the node extraction process, the graph generation algorithm creates the data structure shown in Figure 4.4. Here, one diffusion rectangle is separated into two smaller rectangles and they form two independent electrical nodes. Also, one rectangle in the channel layer is created. The numbers in the figure indicate electrical node numbers assigned. Three electrical nodes are extracted.

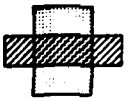


Figure 4.2. Enhancement Mode Transistor

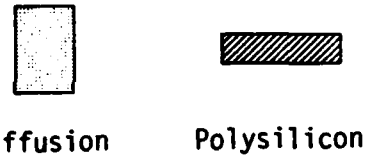


Figure 4.3. Representation of the Transistor

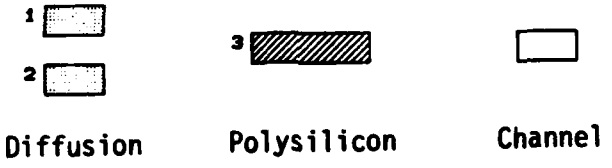


Figure 4.4. Representation of Electrical Nodes

Once the electrical node extraction is completed for each symbolic library element, the rest of the node extraction operation is fairly simple. It is only necessary to take care of connection wires and individual instances of the library symbols. For each instance of a symbol, a separate set of electrical nodes is generated according to the previously extracted electrical node information. In the above example of the enhancement transistor, three electrical nodes are generated for each instance of this symbol. Then, an equivalence relation is constructed among electrical nodes of symbol instances. Two electrical nodes are electrically equivalent if

they are connected by wires, and they are merged into one node.

4.3. Groups and Their Ordering

As discussed in section 3.4.1, vertical and horizontal groups are constructed from an input stick diagram. A group position is calculated by taking an average of coordinates of constituent member elements. The groups are ordered by these group coordinates in ascending order, and are numbered according to this ordering. this numbering is used when separation arcs (simple arcs and dual arcs) are generated. Since all the separation arcs point in the direction of this ordering, the resulting graphs are acyclic.

4.4. Constraints from Design Rules

This section describes the method for generating the minimum distance requirement for two symbols.

For each symbol, we have a data structure maintaining its constituent rectangles for all of the layers. For example, using Mead-Conway type n-mos design rules [MeC80] with supplement [Lyo80], we have the following layers; metal, polysilicon, diffusion, buried cut, cut, implantation. For the purpose of computation, we also have a fictitious layer entry for channel area. Rectangles in the channel layer are automatically generated by the electrical node extraction process.

4.4.1. Calculation of Minimum Distance Requirements

The design rules used here are represented as minimum separation rules. They are stored in a two dimensional table. Using two layers as keys, we can access a minimum distance required between any two layers, if one exists. In order to generate a minimum distance requirement between two symbols, we have to know a relative position between them. This relative position is not necessarily the same as the relative position in an input stick diagram, but it is derived from the group numbers given by the group ordering.

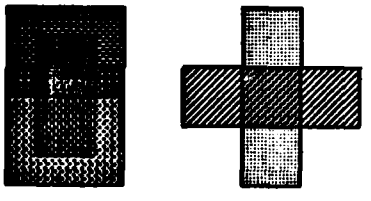


Figure 4.5. Two Non-connecting Symbols

Figure 4.5 represents two symbols located nearby but not connected directly. For each rectangle comprising the symbol, we calculate the extension from the symbol's central point in the direction of the other symbol. In the case of Figure 4.5, the extension of the polysilicon rectangle of the symbol A into the direction of the symbol B (right side) is, $W_a = 2\lambda$. The

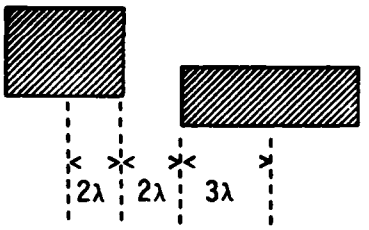


Figure 4.6. Two Polysilicon Rectangles

extension of the polysilicon rectangle in symbol B from symbol's center toward the left is $W_b = 3\lambda$. The design rule requires at least 2λ separation between polysilicon layers, assuming they belong to different electrical nodes. It is represented by $S_{ab} = 2\lambda$. The distance requirement between the central points of symbol A and B, considering only two polysilicon rectangles, is the sum of the above three numbers, $W_a + W_b + S_{ab}$, which is 7λ in our example. The relation between two polysilicon rectangles is represented in Figure 4.6. In this case, the centers of the rectangles coincide with the centers of the symbols to which they belong, but it is not always true. The distance requirement between the centers of two symbols, not the centers of two rectangles, is computed. In order to calculate the minimum distance requirement between symbol A and symbol B, we have to perform a similar computation for every combination of their constituent rectangles. For example, considering a polysilicon rectangle of symbol A and a diffusion rectangle (there are two rectangles but either one is fine) of symbol B, we have $W_a = 3\lambda$, $W_b = 1\lambda$, and $S_{ab} = 1\lambda$. Therefore we have a 5λ separation requirement. If there is no separation requirement rule between the two layers involved, for example between metal and diffusion layer, or they belong to the same electrical node, we do not produce any separation requirement. We take the maximum of these separation requirements calculated between every possible pair of constituent rectangles of the two symbols. In the above example, the final design rule requirement is 7λ.

4.4.2. Connected Elements without Freedom of Movement

In the previous section, we described the general method of computing a minimum distance requirement between two symbols. This method produces an over-constraining value in certain cases. Consider the situation illustrated in Figure 4.7. Here, a horizontal wire L connects a symbol A and a symbol B. Since two elements are connected by a single wire, we generate a simple arc between corresponding nodes in the horizontal graph G_x. The arc is pointing from a node corresponding to A to a node corresponding to B. If the weight of the arc is calculated by the method described, it will be 7λ. The half width of the polysilicon rectangle of symbol A is 3λ, the half width of the diffusion rectangle of symbol B is 3λ, and the minimum space requirement between polysilicon and diffusion layers is 1λ, yielding 7λ as the minimum distance requirement. However, this is overly restrictive. Figure 4.8 depicts a situation that the above minimum distance does not allow, even though the layout is perfectly legal. The constraint between the polysilicon rectangle and the diffusion rectangle is erroneous in this case. Figure 4.9 shows the positional relation of these two rectangles. They have 1λ of vertical clearance (separation). Because these two elements are directly connected by a single wire L, and because there is no freedom of movement at the connection nodes (diffusion rectangles) of

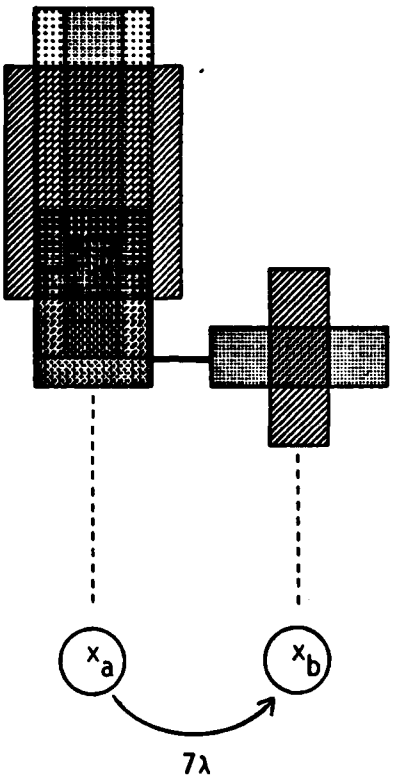


Figure 4.7. Two Connected Symbols

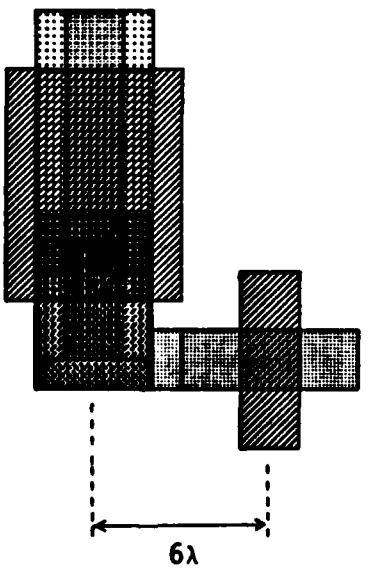


Figure 4.8. Legal Layout

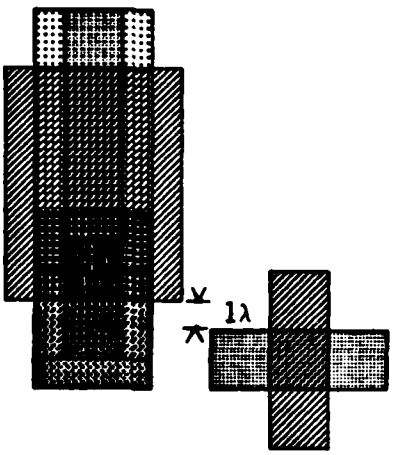


Figure 4.9. Fixed Vertical Clearance

both symbols and wire L, this clearance is always guaranteed. Therefore, as far as the polysilicon layer of symbol A and diffusion layer of symbol B are concerned, we should not generate a minimum distance requirement. The proper minimum distance requirement between symbols A and B is 6λ, resulting from polysilicon-polysilicon considerations.

If we are to generate a minimum distance requirement between two directly connected elements whose relative position in the orthogonal direction is fixed, then we should take into account the separation in the orthogonal direction. For each pair of constituent rectangles, their orthogonal separation is compared to the minimum distance requirement of the design rules. If their separation is greater than or equal to the amount required, then the minimum separation requirement in the original direction is not necessary for that pair of rectangles.

As a special case of this situation, consider the case drawn in Figure 4.10. Here, a horizontal wire L_1 connects a symbol A and a vertical wire L_2. If we do not consider a fixed vertical separation between the diffusion rectangle of symbol A and the polysilicon layer of wire L_2, the resulting minimum distance requirement between two elements is 4λ. This is, of course, over constraining. In this particular case, we do not need to generate any distance requirement between these two elements. Therefore, we do not have any simple arc between the corresponding nodes. This allows much freedom and all the positionings shown in Figure 4.11 are possible.

4.4.3. Choosing Between a Simple Arc and Dual Arcs

As described in Chapter 3, a simple arc is produced if two elements are connected directly by a single wire. Otherwise, a pair of dual arcs is usually generated. A pair of dual arcs introduces one decision variable. A pair of dual arcs is, therefore, much more expensive than a simple arc in terms of computation.

There are certain cases where two elements are not directly connected but we must generate simple arcs in order to maintain non-circularity in the following subgraphs:

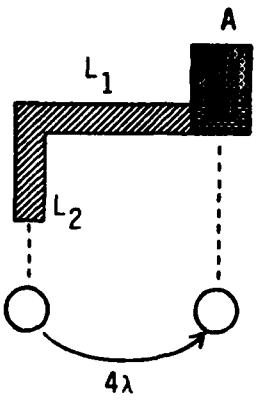


Figure 4.10. Connection between Butting Contact and Poly Wire

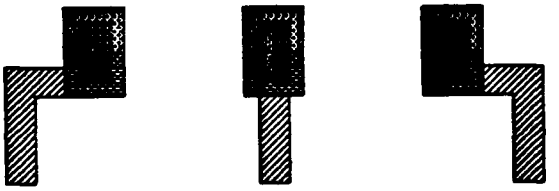


Figure 4.11. Legal Connections

$$G'_x = \{N_x, A_x \bigcup D_x, W_x\}$$

$$G'_y = \{N_y, A_y \bigcup D_y, W_y\}.$$

This restriction prevents generation of positive cycles in the graph G_x and G_y. We present methods for generating either a simple arc or a pair of dual arcs for the following three cases; symbol-symbol, symbol-wire, wire-wire.

Let's introduce some notation. For a given symbol or a wire A, vg(A) represents the vertical group number to which it belongs, and hg(A) represents the horizontal group number. For a horizontal wire segment L_h, vg(L_h) is null, and for a vertical wire segment L_v, hg(L_v) is null.

4.4.3.1. Symbol-Symbol

Assume two symbols A and B are given and they are not directly connected. We need to generate a pair of dual arcs between them if they belong to different horizontal and vertical groups. If they belong to the same group in one of the directions, we should not generate a pair of dual arcs. Moreover in that case, we do not need to generate a simple arc between them unless they are connected to each other by a single wire.

Figure 4.12 represents symbols A and B belonging to the same vertical group but not directly connected.

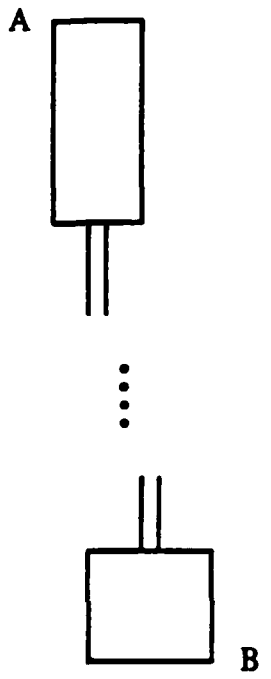


Figure 4.12. Two Symbols in the Same Vertical Group

In this case, the central points of A are located to the left of the central points of B. Even though symbol A is located to the left of symbol B, we should not generate any arc from node x_a to node x_b. If we do, it can form a positive cycle with other connection arcs in the horizontal graph G_x. Furthermore, we do not need any arc from node y_a to node y_b. Symbols A and B being in the same vertical group implies that they are connected by a series of vertical wires without any horizontal wires. We do generate a simple arc between two symbols directly connected by a single wire. Therefore we have a series of simple arcs from node y_b to y_a which guarantees their separation.

If two symbols belong to different vertical groups and different horizontal groups, we generate a pair of dual arcs between them. We generate each arc considering their group numbers, so that the resulting arc points form a node with smaller group number to a node with larger group number.

4.4.3.2. Symbol-Line

Given a symbol A and a wire segment L that are not directly connected, we have to decide whether we should generate a pair of dual arcs or a simple arc. Assume the wire segment L is horizontal without loss of generality. The situation is depicted in Figure 4.13 with two terminating symbols of the wire L.

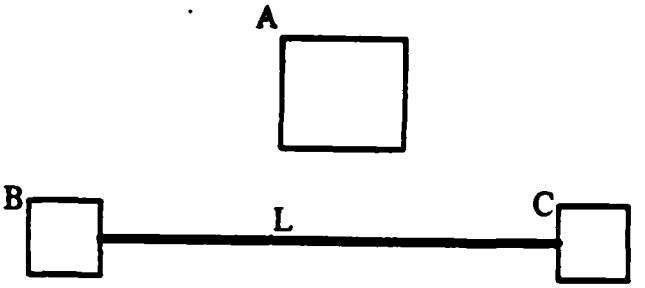


Figure 4.13. Symbol A and Wire L

By the definition of the wire segment, a horizontal wire L does not have an x-coordinate and it is not represented in the horizontal graph G_x. In order to express a situation where symbol A is located entirely on the right side of wire L, we need to use x-coordinate variable x_c of symbol C. That is

$$x_a - x_c \geq d_{ca} \tag{4.1}$$

which represents an arc from node x_c to node x_a. This guarantees that symbol A and wire L will not interfere each other. A similar argument holds for the case where symbol A is located entirely to the left of wire L. We have:

$$x_b - x_a \geq d_{ab} \tag{4.2}$$

which represents an arc from node x_a to node x_b. Both constraints, (4.1) and (4.2), should not exist simultaneously. Since there is a simple arc from node x_b to node x_c, a positive cycle is generated if both constraints (4.1) and (4.2) exist. The situation is shown in Figure 4.14.

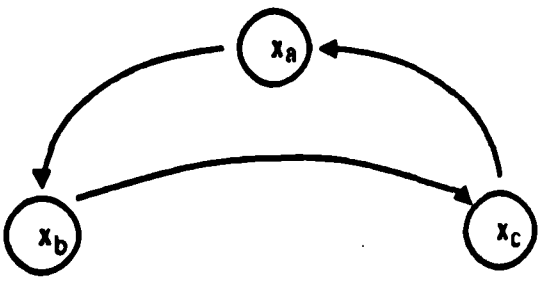


Figure 4.14. Positive Cycle

This positive cycle corresponds to physically impossible conditions. The arc in the vertical direction is straightforward. It points from node y_l to node y_a and represents the inequality:

$$y_a - y_l \geqslant d_{la} \tag{4.3}$$

The method considers the vertical group numbers of the three symbols involved, A, B and C, for choosing a simple constraint or dual constraints. There are three possible cases; one for a simple constraint and two cases for dual constraints.

(1) vg(B) $\leqslant$ vg(A) $\leqslant$ vg(C). In this case, a simple arc is generated. It is in the vertical graph G_y and corresponds to (4.3).

(2) vg(A) < vg(B). Here, A pair of dual arcs is generated. One of them is the arc corresponding to (4.1) and the other corresponding to (4.3).

(3) vg(A) > vg(C). In this case, A pair of dual arcs is generated. One of them is the arc corresponding to (4.2) and the other corresponding to (4.3).

Since vg(B) < vg(C), the above three cases cover all possibilities.

4.4.3.3. Line-Line

Given two wire segments, L_1 and L_2, it is necessary to generate constraints only if they are in the same direction. If one of them is vertical and the other is horizontal, no constraint is necessary between them. The conditions checked are similar to the previous case. Assuming both wires are horizontal, as in Figure 4.15, which shows four symbols directly connected to the wires L_1 and L_2. A vertical component of the dual arc between two wires L_1 and L_2 is straightforward, and it points from node y_{l_2} to node y_{l_1} in the vertical graph G_y. This arc is also shown in Figure 4.15.

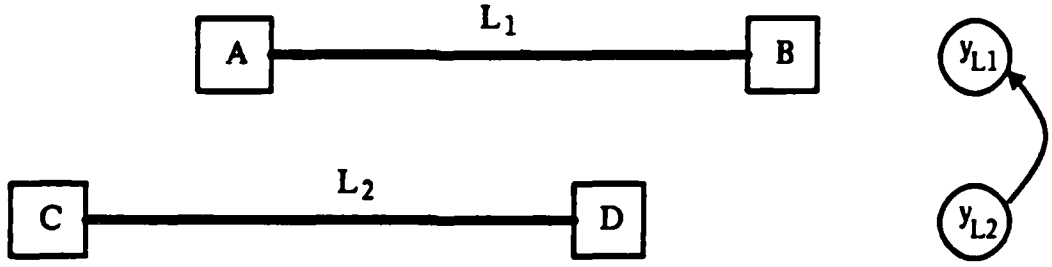


Figure 4.15. Two Wires L_1 and L_2

The method considers the group numbers of symbols A, B, C and D for choice of a single constraint or dual constraints. In the latter case, a proper horizontal arc is also chosen according to their group numbers. The following three cases exist; starting with a case for dual constraints:

(1) vg(B) < vg(C). In this case wire L_1 may be located entirely on the left side of wire L_2. A pair of dual arcs is generated. Its horizontal component corresponds to the following constraint:

$$x_c - x_b \geqslant d_{bc}.$$

(2) vg(D) < vg(A). The wire L_1 may be located entirely on the right side of wire L_2. A pair of dual arcs is generated. Its horizontal component corresponds to the following constraint:

$$x_a - x_d \geq d_{da}.$$

(3) For all other cases, a single vertical arc is generated as a simple constraint. It corresponds to:

$$y_{l_1} - y_{l_2} \geq d_{l_1 l_2}.$$

4.5. Arcs for Connection Requirement

The detailed formulation of the connection requirement is presented in section 3.4.4. The generation of connection arcs is straightforward. However, there is a point to be mentioned regarding the connection requirement among wires. The constraints for the example in section 3.4.4 are shown again.

$$x_{l_1} - x_a \geq d_a$$

$$x_b - x_{l_1} \geq d_b.$$

See Figure 3.10 for an example. They are treated as separation arcs rather than connection arcs. Usually, the separation constraint generation algorithm creates these arcs. However, if the terminating symbols, say symbol A, and wire L_1 have no constraint imposed by the design rules, then special attention is necessary. This condition occurs, for example, in the case that symbol A is a poly-metal contact and wires are polysilicons or metals. The regular constraint generation algorithm does not create any constraint arc between symbol A and wire L_1. The algorithm must generate these arcs as part of the connection requirement, otherwise the integrity of connection between wires L_1 and L_2 is not preserved.

4.6. Elimination of Arcs

In this section, we discuss methods to reduce the number of constraints generated. Let us assume that there are n elements in our layout. If constraints between all possible pairs of elements are generated, there may be up to $\frac{n^2}{2}$ constraints. Each constraint can be either a simple constraint or a dual constraint. However, not all of the constraints are necessary. One reason is that interaction among elements in the layout is local. Another reason is that constraints can affect elements transitively. Therefore some constraints are redundant. As far as the correctness of the final solution is concerned, the redundant constraints do not cause any problem. However, these redundant constraints introduce extra arcs and extra 0-1 variables into the problem, and require unnecessary space and computation.

4.6.1. Use of Compartments

Interactions among elements of a layout are fairly local. There is a substantial number of pairs of elements that do not interact with each other. This interaction among elements is termed *visibility* between two elements [HsP79,SLM82]. Two elements are visible in a placement if there is a straight line segment that intersects both elements without intersecting any other elements. In the process of compaction, elements in a layout shift their position relative to each other. In one placement, a certain pair of elements may not be visible to each

other but they can become visible to each other. On the other hand, there is a pair of elements that never become visible to each other. No constraint should be generated between them. One attempt to capture this locality of interaction is the following idea of *compartments*.

Considering symbols, wires and boundaries as walls, we can regard an empty area surrounded by these walls as a compartment. In constructing compartments, two separate planes are distinguished in our n-mos layout. They are a metal plane and a poly-diffusion plane consisting of other layers (diffusion, polysilicon, implant, cut, buried cut). Figure 4.16 shows the compartment in the poly-diffusion plane for the T-flipflop example. (For the entire stick diagram of the T-flipflop, see Appendix B.)

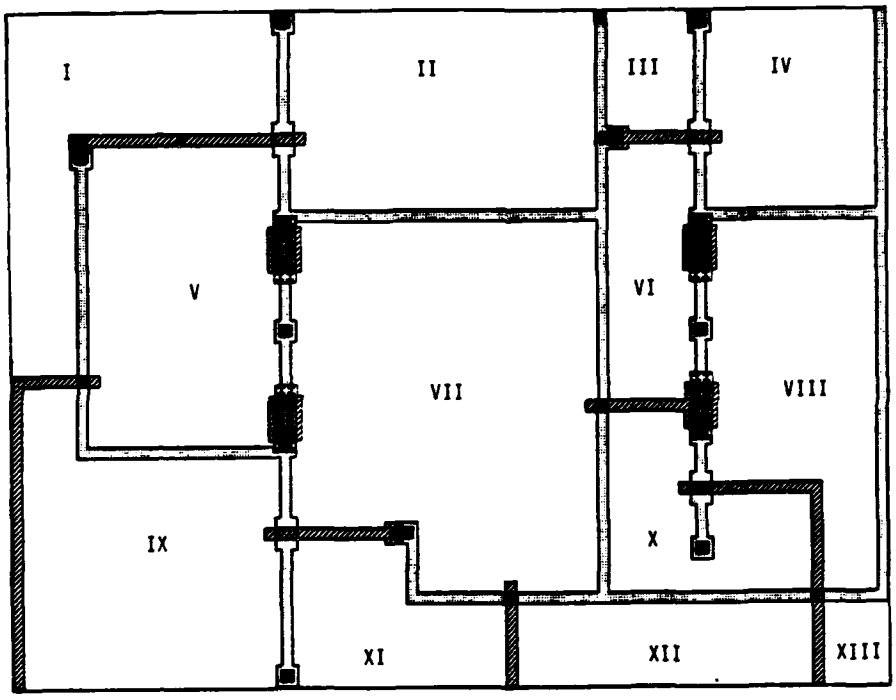


Figure 4.16. Compartment in T-flip-flop Layout

Certain pairs of elements never interact with each other because of intervening wires and symbols between them. They form a wall hiding the elements from each other. There is no need to generate a constraint between these pairs. The compartments are used to find out which pairs of elements interact with each other. Elements or borders that are adjacent to the same compartment can interact with each other through it. If a pair of elements is not adjacent to the same compartment, It is not necessary to generate any constraint between them. Special care is needed if both symbols involved are represented in both planes, that is, they contain metal rectangles and polysilicon or diffusion or both rectangles (as butting contact). If they are adjacent to the same compartment in one plane but not in the other, then no constraint is generated. In this case, they interact with each other only when they interact in both planes.

The compartments are constructed by a scan line method (a pond and island algorithm). The technique is used in computer vision [BaB82, p151] and is also applied for node extraction from mask layout of integrated circuits [Bak80, BaT80]. The layout is represented by a two dimensional bit array, each bit representing a lambda square. If a bit is one, there is some circuit located on the corresponding position of the layout. If it is zero, there is no element in that position. These algorithms process a single bit (picture element) at a time, and they distinguish each empty space, which is a group of connected zero bits, as a compartment. The data structure used for representing a layout is not a bit array but a set of rectangles. Therefore, to speed up the procedure, the idea of an event list [BHH80] is used. Each event represents a starting edge and a trailing edge of each rectangle of concerning layers. It identifies isolated empty spaces separated by symbols and wires. This operation is performed twice; once considering only metal rectangles and a second time considering all other rectangles.

4.6.2. Reduction by Transitive Closure

Even though compartments are used to reduce unnecessary arcs, a fair number of redundant arcs may still remain. The another group of redundant arcs results because constraints between elements propagate through more than one arc. Consider constraints represented by a graph shown in Figure 4.17.

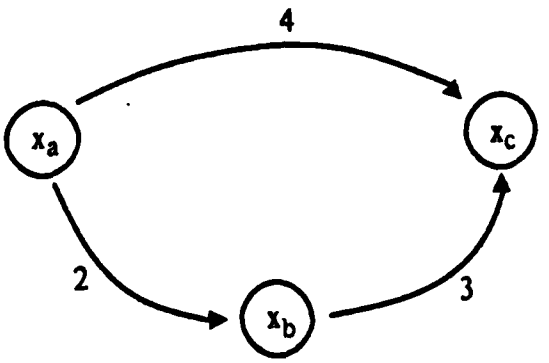


Figure 4.17. Transitivity of Arcs

There are three horizontal coordinates of three elements represented by x_a, x_b and x_c. Assume that there are three separate constraints as represented by three arcs in Figure 4.17. Also assume that arcs $<x_a, x_b>$ and $<x_a, x_c>$ belong to $A_x \bigcup R_x$ and not to D_x. The weights of the three arcs $<x_a, x_b>$, $<x_b, x_c>$, and $<x_a, x_c>$ are 2λ, 3λ, and 4λ, respectively, as shown in figure 4.17.

The sum of weights of arcs $<x_a, x_b>$ and $<x_a, x_c>$ is 5λ. Therefore, the elements A and C are required to be separated by 5λ. Since these two arcs belong to $A_x \bigcup R_x$, they are always present. Therefore, 5λ separation is always required. This requirement of 5λ is larger than the weight of arc $<x_a, x_c>$ which is 4λ. The minimum separation requirement between elements A and B, calculated directly, is 4λ, but they are forced to be separated by 5λ by two other arcs. The arc $<x_a, x_c>$, therefore, becomes redundant in this case, and it can be

eliminated.

Now, we consider a case where arc $<x_a, x_c>$ belongs to D_x. Let $e_i^x = <x_a, x_c>$ and the i-th dual pair be $<e_i^x, e_i^y>$. Since e_i^x is redundant and its requirement is always satisfied, we do not need to consider a requirement represented by e_i^y at all. Therefore, the i-th pair of dual arcs can be eliminated altogether. Since each dual pair introduces a 0-1 variable and is potentially costly in computation, it is very helpful to eliminate one.

No arc belonging to R_x is eliminated by this transitivity. This follows directly from the construction of graph G_x. Remember all arcs in R_x connect two nodes in the same vertical group and adjacent to each other. However each arc in $A_x \bigcup D_x$ connects two nodes that belong to different vertical groups, and always points toward groups with larger group (ordering) number. Therefore there is no alternate path from a tail node of an arc in R_x to its head node. Similar arguments hold for vertical graph G_y.

Let us summarize the above discussion. Let u and v be two nodes in a graph. Assume that there is an arc <u, v>. Assume further that there is a path from u to v consisting of two or more arcs in set $A \bigcup R$. An arc <u, v> can be eliminated if its weight is smaller than or equal to the sum of weights of all arcs in the path. Moreover, if arc <u, v> is a part of dual arcs, then one can eliminate its counterpart in the graph for the other direction. No arc of set R is eliminated by this operation because of the rules governing its construction.

CHAPTER 5

Compaction Algorithm

5.1. Introduction

The compaction algorithm is formulated in this chapter. The algorithm is based on a branch and bound method and a longest path algorithm. At each stage of branch and bound operation, the longest path algorithm is executed. However, the compaction algorithm starts by computing an initial solution by a heuristic method. The branch and bound method attempts to improve this initial solution. The better the initial solution is, the better the branch and bound search performs.

5.2. Upper and Lower Subgraphs

In order to formulate the algorithm, we introduce four subgraphs of G_x and G_y. We also introduce another 0-1 variable μ_i for each pair of dual arcs $<e_i^x, e_i^y>$. The subgraphs are called upper graphs and lower graphs of the horizontal and vertical graphs. They are used to compute an upper bound and a lower bound of the layout area.

For a given decision vector δ^0, the upper graphs are defined as follows:

$$g_x^u(\delta^0) = \{N_x, E_x^u(\delta^0), W_x\}$$

$$g_y^u(\delta^0) = \{N_y, E_y^u(\delta^0), W_y\} .$$

$g_x^u(\delta^0)$ is a subgraph of G_x and $g_y^u(\delta^0)$ is a subgraph of G_y. The arcs in $g_x^u(\delta^0)$ are all of the arcs in $A_x \bigcup R_x$ and the subset of arcs in D_x whose decision variable δ_i^0 equals 0. That is

$$E_x^u(\delta^0) = R_x \bigcup A_x \bigcup D_x^u(\delta^0)$$

where

$$D_x^u(\delta^0) = \{e_i | e_i \in D_x \text{ and } \delta_i^0 = 0\} .$$

Similarly,

$$D_y^u(\delta^0) = \{e_i | e_i \in D_y \text{ and } \delta_i^0 = 1\} .$$

By the above definition one of the arcs in each dual pair is always represented in either $g_x^u(\delta^0)$ or $g_y^u(\delta^0)$. If all the inequalities associated with the arcs in $g_x^u(\delta^0)$ and $g_y^u(\delta^0)$ are satisfied, the resulting vector $\mathbf{z} = <\mathbf{x}, \mathbf{y}, \delta>$ is feasible, and it represents a legal layout.

We now explain the 0-1 vector μ. Originally a vector δ is given temporary values by a heuristic algorithm. These are initial estimates for the optimal solution. We call the value of δ_i permanent after it is corrected or endorsed by the branch and bound part of the algorithm. The 0-1 vector μ represents which components of decision vector δ are temporary or permanent. Let $\mu_i = 0$ mean that the value of δ_i is temporary. If the value of δ_i is permanent, it will not be changed in a future branch and bound operation by moving down the search tree. At the root of a branch and bound search tree, all values of components of δ^0 are temporary. Therefore, all components of μ^0 are 0's.

For given vectors δ^n and μ^n, the lower subgraphs are defined as:

$$g_x^l(\delta^n, \mu^n)=\{N_x, E_x^l(\delta^n, \mu^n), W_x\}$$

and

$$g_y^l(\delta^n, \mu^n)=\{N_y, E_y^l(\delta^n, \mu^n), W_y\} .$$

$g_x^l(\delta^n, \mu^n)$ is a subgraph of G_x and $g_y^l(\delta^n, \mu^n)$ is a subgraph of G_y. The arcs in $g_x^l(\delta^n, \mu^n)$ are all of the arcs in $A_x \bigcup R_x$ and the subset of D_x whose decision values are permanently decided to be 0. More formally,

$$E_x^l(\delta^n, \mu^n)=A_x \bigcup R_x \bigcup D_x^l(\delta^n, \mu^n)$$

$$D_x^l(\delta^n, \mu^n)=\{e_i | e_i \in D_x, \delta_i^n= 0 \text{ and } \mu_i^n= 1\} .$$

Similarly for the vertical direction,

$$D_y^l(\delta^n, \mu^n)=\{e_i | e_i \in D_y, \delta_i^n= 1 \text{ and } \mu_i^n= 1\} .$$

In the lower subgraphs, not all of the dual arcs are represented. Only a permanently decided pair has its members represented either in $g_x^l(\delta^n, \mu^n)$ or in $g_y^l(\delta^n, \mu^n)$. Therefore, even if all of the inequalities associated with the arcs in $g_x^l(\delta^n, \mu^n)$ and $g_y^l(\delta^n, \mu^n)$ are satisfied, the resulting $\mathbf{z}=<\mathbf{x}, \mathbf{y}, \delta>$ may not be feasible.

There are two special nodes, a source and a sink, in each of these subgraphs. They are denoted by s_x^u and t_x^u for the upper subgraph $g_x^u(\delta^0)$. Similar notation is used for other subgraphs. $(t_x^u- s_x^u)$ represents an upper bound of the width of the layout and $(t_x^l- s_x^l)$ represents its lower bound.

5.3. Mechanism of the Branch and Bound Method

Here, we describe the overall mechanism of the branch and bound algorithm. The purpose of the branch and bound is to remove a dual arc from a longest path of the upper graphs and enter its counterpart in the other direction. At each step of a branch and bound, the algorithm constructs a longest path spanning tree rooted at a source node for each of the four subgraphs. From the spanning trees of the upper subgraph, the following two kinds of information are obtained. The first is coordinates of symbols and lines of a layout. In this layout all of the elements are packed to the left and to the bottom as much as possible satisfying currently active constraints. The second is sets of arcs which make up the longest paths in this layout. The spanning trees of the lower subgraph are used to calculate lower bounds on the width and height of the layout.

In order to explain a branching operation at some node in the branch and bound search tree, we assume the following conditions without loss of generality

(1) $\mu_i=0$, i.e., a decision made on i-th dual pair is temporary.

(2) $\delta_i=0$, i.e., an active arc in the i-th pair, $<e_i^x; e_i^y>$, is its x-component e_i^x.

(3) The i-th x-arc e_i^x is on the longest path in the upper graph $g_x^u(\delta^n)$.

This situation in the branch and bound search tree is depicted in Figure 5.1. In order to shorten the length of the horizontal longest path, the i-th x-arc will be removed. Then the i-th y-arc will be entered into the graph $g_y^u(\delta^n)$ and $g_y^l(\delta^n, \mu^n)$. A necessary operation is to set δ_i and μ_i to one in this example. This operation is called a correction of the initial estimate, and this branch in the search tree is called a correction branch. The correction operation is defined as

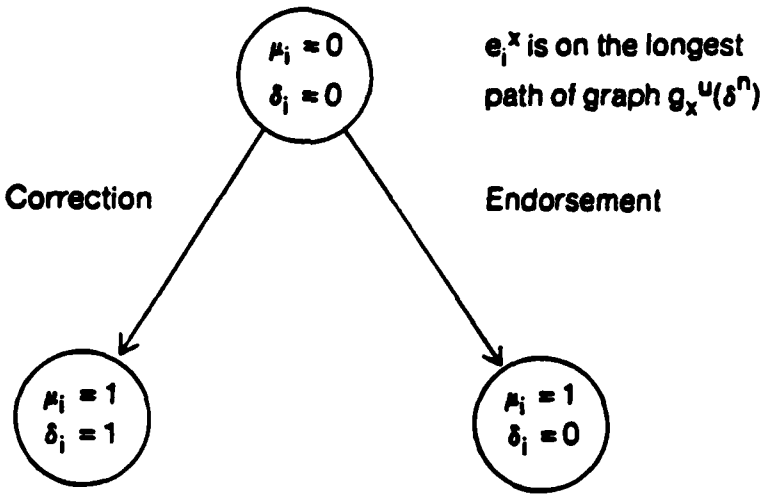


Figure 5.1. Single Branching

$\delta_i := \neg\, \delta_i$

$\mu_i := 1$.

The correction operation modifies three subgraphs; two upper graphs and one lower graph. The other branch is called an endorsement branch. An operation associated with this branch is to set μ_i to one. This endorses the initial temporary decision on i-th pair of dual arcs. The endorsement operation modifies only one subgraph, one lower graph. Since the purpose of branching is removal of a dual arc from a longest path, a correction branch is always explored first. An endorsement branch is explored only after the corresponding correction branch is completely searched.

After a branching is made, some of the longest paths must be reconstructed. In the correction branch, it is necessary to update three spanning trees. The arc e_i^x is removed from the graph $g_x^u(\delta^n)$. On the other hand, arc e_i^y is entered in both graphs $g_y^u(\delta^n)$ and $g_y^l(\delta^n, \mu^n)$. The spanning trees associated with these graphs must be updated. In the endorsement branch, the only change is the addition of the arc e_i^x into the graph $g_x^l(\delta^n, \mu^n)$, and only one spanning tree has to be updated.

The resulting subgraphs are different from their predecessor subgraphs only by one arc. The predecessor subgraphs are the ones associated with the parent node in the search tree. For example, the graph $g_x^u(\delta^{n+1})$ is different from the graph $g_x^u(\delta^n)$ by removal of a single arc e_i^x in the above example of correction branch. The updating of the longest path spanning trees can be done with fewer steps using the existing longest path spanning tree at hand rather than constructing an entirely new one.

5.4. Pruning Conditions

Here, we consider the conditions under which one can prune a certain subtree in the branch and bound search tree.

At each node in the branch and bound search tree, two functions are evaluated; $\bar{f}$ and $\underline{f}$. Let us denote an arbitrary current node by η. Then the situation is shown in Figure 5.2.

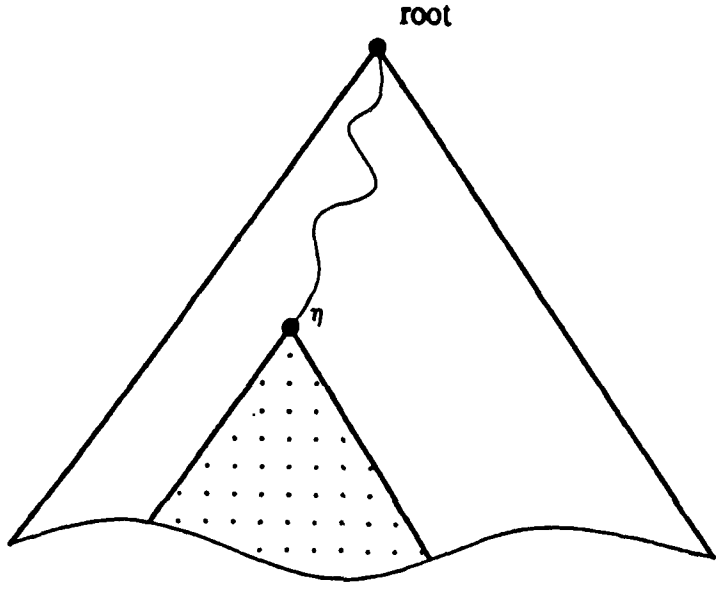


Figure 5.2. Branch and Bound Search Tree

These two functions represent a local upper bound and a lower bound of an area occupied by a layout.

$\bar{f}=(t_x^u - s_x^u)(t_y^u - s_y^u)$ upper bound

$\underline{f}=(t_x^l - s_x^l)(t_y^l - s_y^l)$ lower bound .

The value of $\underline{f}$ increases monotonically as a search process goes deeper into the search tree. This is because the number of 1's in the vector μ is equal to the depth of the node η from the root, and its components are never reset from 1 to 0 by going down.

The global best upper bound f^* of $f(\mathbf{x},\mathbf{y})$ is maintained. This upper bound is updated at each time the local upper bound $\bar{f}$ becomes smaller than the value of f^*. This value represents the currently known best solution. It is also necessary to maintain vectors $\mathbf{x}$, $\mathbf{y}$ associated with the global upper bound f^*. They represent actual coordinates of the symbols and lines of the currently known best layout.

An optimal area, f^η, obtainable in the subtree rooted at the node η is bounded by the functions $\bar{f}$ and $\underline{f}$ calculated at node η:

$$\underline{f} \leqslant f^{\eta} \leqslant \bar{f} .$$

Note that the bounds calculated at the node η hold for the whole subtree under η.

We now present the following two pruning conditions:

$$\begin{aligned} &\underline{f} \geqslant f^{*} \\ &\underline{f} = \bar{f} . \end{aligned} \tag{5.1}$$

Condition (5.1) indicates that any solution which could be found in the subtree rooted at node η is greater than or equal to the currently known best solution. Therefore, we can cut this subtree from the search tree.

Condition (5.2) shows that the values of the upper bound and the lower bound equal each other. There will be no further improvement in the subtree under this node η. Actually, condition (5.2) is absorbed in condition (5.1), since, as soon as an local upper bound $\bar{f}$ is evaluated, it is compared with the global upper bound f^*. If $\bar{f} < f^*$, the global upper bound is updated to a value $\bar{f}$. Therefore, it is only necessary to check condition (5.1).

5.5. Construction of Longest Path Spanning Tree

The graphs do not have a positive cycle because of their construction. Therefore, the Bellman-Ford successive approximation method [Law76] can be used for determination of the longest path. Its run time is $O(n^3)$ where n is the number of nodes in the graph. In order to find the longest path, a longest path spanning tree is constructed. The spanning tree also provides the coordinates of all elements in the layout.

Since our graphs are in very restricted form, the Bellman-Ford successive method is not used in its most general form. Our graphs are almost acyclic. The cycles appear as pairwise loops, which represent connection between symbols and lines. Therefore, one can use an algorithm which is a slight modification of the longest path algorithm on an acyclic graph. The runtime of the acyclic graph algorithm is linear in the number of arcs. First, the acyclic algorithm is outlined. In this algorithm, a topological sort is performed to order the nodes,

$$z_1, z_2, z_3, \ldots, z_i, z_{i+1}, \ldots, z_n$$

so that there is no arc from node z_i to z_j, if $i \geqslant j$. Then, the following computation is performed:

$$z_1 := 0$$

$$z_j := \max_{k<j} \{z_k + w_{kj}\}, \quad j = 2, \ldots, n.$$

For each z_j, we only need to consider z_k such that $<z_k, z_j>$ is an arc in the graph.

It is necessary to modify this algorithm so that it takes care of pairwise loops created by arcs belonging to the set R_x and R_y. The nodes in our graphs cannot be ordered as in the case of an acyclic graph. There is not a partial ordering among nodes in our graph. However, there is a total ordering among groups of nodes. Within these groups, nodes are connected by arcs in the set R_x and R_y.

The modified algorithm is explained in the following. Let the i-th group be denoted by Q_i. Construct the following ordering of nodes, which is based on a total ordering of groups:

$$Q_1, Q_2, \ldots, Q_i, \ldots, Q_n$$

where

$Q_1 = \langle z_1 \rangle$

:

$Q_i = \langle z_i, \ldots, z_j \rangle$

$Q_{i+1} = \langle z_{j+1}, \ldots, z_k \rangle$

:

$Q_n = \langle z_n \rangle$.

Within a group, order the nodes according to the positioning in the input layout. Node z_1 is a source node (s_x or s_y); node z_n is a sink node (t_x or t_y). They form a group by themselves. Perform the acyclic algorithm for each node from left to right in the above ordering until encountering the end of the group. In this step, consider only arcs in set $A_x \bigcup D_x$ or $A_y \bigcup D_y$. After having computed values of all nodes in one group, traverse all nodes in this group from left to right, checking whether conditions represented by arcs in set R are satisfied or not. That is done by checking if the following two inequalities:

$$z_j \geq z_i + w_{ij}$$

$$z_i \geq z_j + w_{ji}$$

are satisfied or not. If there is an inequality that is not satisfied, satisfy it by updating a value of the appropriate node. It is only allowed to increase a value, not to decrease it. Then, backtrack from the updated node to the left to take care of the ripple through of the new increase. Backtracking terminates upon finding a pair of satisfied inequalities. Then, resume the original process from the node which originally caused the backtracking. Reaching the last node in a group implies that the positioning of elements within that groups is consistent. Then, proceed to the next group to perform the acyclic algorithm, and the whole process repeats.

5.6. Heuristic Method for the Initial Estimate

The method for finding an initial estimate is basically iterative. Initially, all of the dual arcs in D_x and D_y are activated. At each step of the iteration, longest paths in the horizontal graph and in the vertical graph are constructed. Then, the dual arcs that contribute to one of the longest path but not the other are removed by setting components of vector δ properly. As soon as some dual arcs are removed their counterparts in the other direction will never be removed. The algorithm iterates until no more arcs are removed from either graphs. At termination, there are still some dual arcs whose components are represented in both graphs. They consist of two groups of arcs. The first group is the dual arcs that contribute to the longest path of both directions. The second group is the dual arcs that do not contribute to either of the longest paths. For each pair of dual arcs in both groups, compare weights associated with a vertical arc and a horizontal arc. Then, choose the arc with smaller weight and remove the arc with larger weight.

We present the algorithm more formally using an algorithmic language. In the following presentation, we introduce two subgraphs $g_x^h(\delta)$ and $g_y^h(\delta)$, and assume $\delta_i \in \{-1, 0, 1\}$. $\delta_i = -1$ means that the i-th dual arcs are represented in both subgraphs $g_x^h(\delta)$ and $g_y^h(\delta)$ Initially, $\delta_i = -1$ for all i. Then

$$g_x^h(\delta) = \{N_x, E_x^h(\delta), W_x\}$$

$$E_x^h(\delta) = R_x \bigcup A_x \bigcup D_x^h(\delta)$$

$$D_x^h(\delta) = \{e_i \mid e_i \in D_x, \delta_i = -1, \text{ or } \delta_i = 0\}.$$

Similarly, for the vertical direction:

$g_y^h(\delta)=\{N_y, E_y^h(\delta), W_y\}$

$E_y^h(\delta)=R_y \bigcup A_y \bigcup D_y^h(\delta)$

$D_y^h(\delta)=\{e_i | e_i \in D_y, \delta_i=-1, \text{ or } \delta_i=1\}$.

The heuristic algorithm for the initial estimate is as follows:

Set $\delta_i \leftarrow -1$ for all i $\in$ I.
Repeat

Find a longest path in $g_x^h(\delta)$
Let $P_x \leftarrow \{e_i | e_i \in D_x, \delta_i=-1, \text{ and } e_i \text{ is part of the longest path of } g_x^h(\delta)\}$
Find a longest path in $g_y^h(\delta)$
Let $P_y \leftarrow \{e_i | e_i \in D_y, \delta_i=-1, \text{ and } e_i \text{ is part of the longest path of } g_y^h(\delta)\}$
$Q_x \leftarrow P_x - (P_x \bigcap P_y)$
$Q_y \leftarrow P_y - (P_x \bigcap P_y)$
For all $e_i \in Q_x$ do $\delta_i \leftarrow 1$ (* drop x-component *)
For all $e_i \in Q_y$ do $\delta_i \leftarrow 0$ (* drop y-component *)

Until $(Q_x = \varnothing)$ and $(Q_y = \varnothing)$
For all $<e_i^x, e_i^y>$ such that $\delta_i=-1$ do

if (weight of e_i^x) $\geqslant$ (weight of e_i^y) then
$\delta_i \leftarrow 1$
else
$\delta_i \leftarrow 0$

5.7. Choice of Branching Arcs

In the branch and bound method previously described, one arc is chosen arbitrary for branching. It is possible to speed up the search process tremendously by choosing the most promising arc. We describe a possible method for choosing a promising arc.

5.7.1. Estimating the Change in the Layout Area

Our goal is to minimize the layout area $f = t_x^u \cdot t_y^u$. For an example, let us assume that a branching is done by setting δ_i from 0 to 1. That is, an x-arc e_i^x is removed and a y-arc e_i^y is entered. Let Δx be a resulting change (decrease) in the width of the layout and Δy be a resulting change (increase) in the height. Notice that $\Delta x \leqslant 0$ and $\Delta y \geqslant 0$. Then new area f' of the layout is

$$f' = (t_x^u + \Delta x)(t_y^u + \Delta y)$$
$$= t_x^u \cdot t_y^u + t_x^u \cdot \Delta y + t_y^u \cdot \Delta x + \Delta x \cdot \Delta y$$
$$= f + t_x^u \cdot \Delta y + t_y^u \cdot \Delta x + \Delta x \cdot \Delta y$$

where f is the current area. The last two terms in the above expression are non-positive. Usually, the exact value of Δx and Δy is not known and their values have to be estimated. Let us define a function:

$$\Delta f = t_x^u \cdot \Delta y + t_y^u \cdot \Delta x + \Delta x \cdot \Delta y .$$

Then, the arc that minimizes the value of the above function Δf is chosen as the next branching arc.

One possible method of estimating the values of Δx and Δy is as the following. In order to avoid confusion, notation $\dot{x}_i$ is used to represent the value of node x_i in this section. Let arc e_i^x be $<x_j, x_k>$ and w_{jk}^x be its weight. Then we consider all incoming arcs to node x_k except e_i^x. See Figure 5.3.

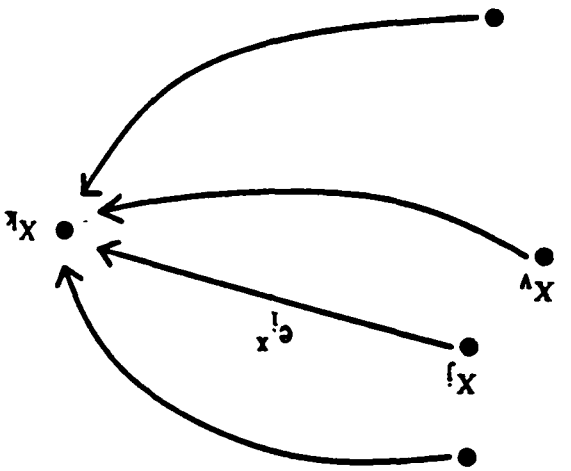


Figure 5.3. Incoming Arcs to Node x_k

For each such arc $<x_v, x_k>$, we calculate the sum of the values of its tail node and its weight, $\dot{x}_v + w_{vk}^x$. We choose the largest of these sums and let it be $\ddot{x}_k$. That is

$$\ddot{x}_k = \max_v \{\dot{x}_v + w_{vk}^x\} .$$

$\ddot{x}_k$ will be the new value of node x_k after removal of e_i^x. The difference

$$\Delta x = \ddot{x}_k - \dot{x}_k$$

is the bound of possible change (decrease) in the width of the layout. Due to the construction of the graph and definition of values of nodes in the graph, the value Δx is always non-positive when x-arc is removed. In practice, it is not necessary to perform the above calculation as an extra operation. This is one of the operations performed for constructing a longest path spanning tree rooted at the source of the graph. It is necessary to retain two largest values at each node. The second largest value at node x_k is the above value $\ddot{x}_k$.

We now discuss a method for estimating the value of Δy. Let a newly introduced arc be $e_i^y = <y_j, y_k>$ and its weight be w_{jk}^y. Then a possible new value of node y_k is computed as:

$$\ddot{y}_k = \dot{y}_j + w_{jk}^y .$$

If $\ddot{y}_k \leq \dot{y}_k$, then the value of node y_k will not be changed by the introduction of a new arc e_i^y. On the other hand, if $\ddot{y}_k > \dot{y}_k$ then the estimate of the increase of a height of a layout is

$$\Delta y = \ddot{y}_k - \dot{y}_k .$$

Actually, this is an upper bound on the possible increases in the height of the layout.

If one is willing to perform further computation, it can be checked if the increase at the node y_k propagates or not. For this purpose, for each outgoing arc from node y_k, the following is calculated:

$$\ddot{y}_z = \ddot{y}_k + w_{kz}^{y} .$$

where y_z is a head node of such an outgoing arc. If $\ddot{y}_z \leqslant \dot{y}_z$, then the increase of $\Delta y = \ddot{y}_k - \dot{y}_k$ at node y_k is absorbed by slack space associated with arc $<y_k, y_z>$. If this is true for all outgoing arcs, then the increase of Δy is completely absorbed by slack space and the resulting increase in layout area is zero. If $\ddot{y}_z \geqslant \dot{y}_z$ at any node y_z, then at least part of the increase, Δy, propagates along an arc $<y_k, y_z>$. The amount of propagation is $\ddot{y}_z - \dot{y}_z$. Let the largest of $\ddot{y}_z - \dot{y}_z$ over all such arcs be $\Delta' y$. Let $\Delta' y$ be defined as

$$\Delta' y = \max_z \{\ddot{y}_z - \dot{y}_z\}$$

Then, $\Delta' y \leqslant \Delta y$. Therefore, $\Delta' y$ is possibly a tighter and better bound than Δy for an increase of height of a layout. In this method of computation, the propagation of increases may result in a tree structure. If this tree structure actually reaches the sink of the graph, we know the exact amount of increase. However, this is a fairly involved and inefficient computation and we have to make a trade off between an accuracy of estimation of true value of Δy and the amount of computation. In the following section, we will present a more elegant and efficient method to obtain exact value of Δy.

5.7.2. Maximum Non-increasing Coordinate

The above method should not be used for computing an exact value of Δy because it repeatedly computes the same information for each candidate arc for branching. Instead, the maximum non-increasing coordinate is calculated for each node. The maximum non-increasing coordinate is defined as the maximum value that a node can take without increasing the value of a sink node. This calculation is performed only once for each step of branching. The algorithm is basically the same as the one used for construction of a longest spanning tree rooted at a source node. Let us denote the maximum non-increasing coordinate associated with node z_i to be $\bar{z}_i$. For all nodes in the longest path form a source node to a sink node:

$$\bar{z}_i = \dot{z}_i .$$

Specifically, for sink node z_t, which forms a root node:

$$\bar{z}_t = \dot{z}_t$$

One performs the algorithm described in section 5.5 in descending ordering of the groups. The operation performed at the acyclic part of the algorithm is replaced by

$$\bar{z}_t := \dot{z}_t$$

$$\bar{z}_i := \min_{j>i} \{\bar{z}_j - w_{ij}\}, \quad i = n-1, \ldots, 1 .$$

Also, the updating operation performed within the group is done by decreasing the value $\bar{z}_i$ and not by increasing it. A difference $\bar{z}_i - \dot{z}_i$ is the number that can be called *cumulative slack space*. A value of node z_i can be increased by this amount or less without increasing a layout dimension. Now, it is possible to calculate the exact amount of increase in the dimensions of a layout. Using the example of the previous section:

$$\Delta y = \begin{cases} \ddot{y}_i - \bar{y}_i & \text{if } \ddot{y}_i > \bar{y}_i \\ 0 & \text{otherwise.} \end{cases}$$

5.7.3. Merit of the Algorithm

Evaluating function Δf is done only for arcs that are located on the longest path from a source to a sink in either graph. Removal of one of them may decrease the layout area. Removal of arcs that are off the longest path does not decrease the overall dimensions of the layout. If some pairs of arcs never appear on either of the longest paths, then the above calculation is never done for these arcs. Moreover, they are never part of the branch and bound operation. This is an important advantage of the branch and bound algorithm presented here. Redundant arcs (ones not eliminated) and non-critical dual arcs are never considered in the branch and bound operation. The problems solved usually have a large number of integer variables but the majority of their values are set once by the initial heuristic routine and are never reset again, because they never participate in the branch and bound operation.

CHAPTER 6

Application and Extension of the Method

6.1. Introduction

In this chapter, the concepts and the algorithms developed in the previous chapters are examined for further generalities and possibilities. Especially, the goal directed compaction is described.

6.2. User Supplied Goals

One of the strong points of the compaction algorithm is that it can incorporate certain user supplied goals and requirements as constraints. In most cases, the algorithm does not need any modification. The ability to incorporate user supplied goals is also important if the algorithm is consolidated into a system for automatic IC layout or a silicon compiler. In that case, goals can be supplied by not only a human but also a higher level routine in an hierarchical layout system. In this section, we discuss two kinds of user supplied goals and constraints which can be incorporated into the compaction algorithm elegantly. The first is related to the positions of connection points of the cell to the outside. The second is related to the final shape of a compacted cell.

6.2.1. Connection Points

Laying out a single basic cell and duplicating it to create a final unit is common practice in the design of integrated circuits. A dynamic shift cell is a good example. A single basic cell is shown in Figure 6.1.

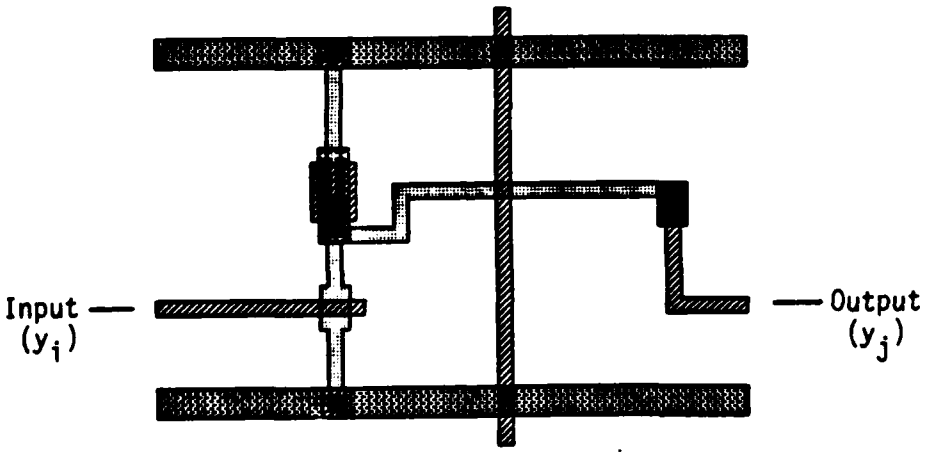


Figure 6.1. A Shift Cell

To create an eight bit dynamic shift register, the designer duplicates the cell sixteen times in the horizontal direction. This style of design reduces the designer's work and helps to produce a more regular structure in an IC layout. For the cell to be useful, positions of the input and output of the shift cell must agree with each other. This constraint can be expressed in our scheme easily. Let node y_i represent a vertical position of the input wire and node y_j represent that of the output wire as shown in Figure 6.1. The constraint can be incorporated by introducing a pair of connection arcs between node y_i and node y_j. It is depicted in Figure 6.2. By introducing the pair, we are merging two horizontal groups to which node y_i and y_j belong. There may be certain constraint arcs that should not be generated because of this merging.

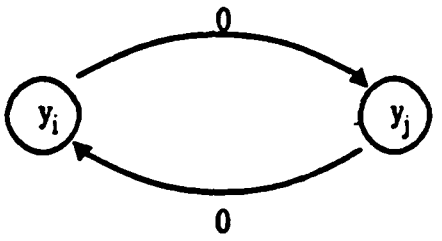


Figure 6.2. Arcs for Connection Points

If a connection point is fixed, the algorithm should simply fix the value of that node. Let node y_k represent a vertical position of some connection point that is fixed at a fixed position c as shown in Figure 6.3.

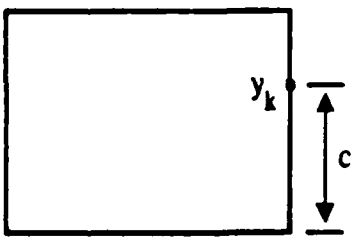


Figure 6.3. Fixed Height Points

Then the value of y_k must be initialized to c. At a longest path spanning tree construction step, we do not update the value of y_k. However, all the calculations of values of nodes are performed normally. If the newly calculated value of y_k exceeds c, that branch of the search tree is immediately fathomed. Also, there is a possibility that the longest path that determines the dimensions of layout extends from node y_k to the sink node, rather than from the source node to the sink node. Another slight modification must be made to the heuristic algorithm for an initial solution. The algorithm must generate a feasible solution for an initial estimate. If the value of y_k is greater than c, it removes the arcs that are on the longest path from the source node to node y_k. If the value of y_k is smaller than or equal to c, it executes as described in

section 5.6. If the algorithm fails to achieve a value smaller than c, the goal given by a user is inconsistent.

It is possible to incorporate the fixed constraints in a more general form. The longest path algorithm used for constructing a spanning tree must be replaced by a more general algorithm. So far all the constraints except connection constraints are constraints of the lower-bound type, that is

$$x_i - x_j \geqslant c. \tag{6.1}$$

It is very useful if the compaction algorithm can handle the spacing requirements expressed by the equality or the upper-bound type constraints between two mask elements. An example of these constraints is as the following:

$$x_i - x_j \leqslant c. \tag{6.2}$$

$$x_i - x_j = c. \tag{6.3}$$

This allows the user more control over the compaction process. Liao and Wong developed the algorithm that can handle these constraints in the context of one-dimensionsal compaction [LiW83]. In their method, the upper-bound constraints can be represented by the graph by

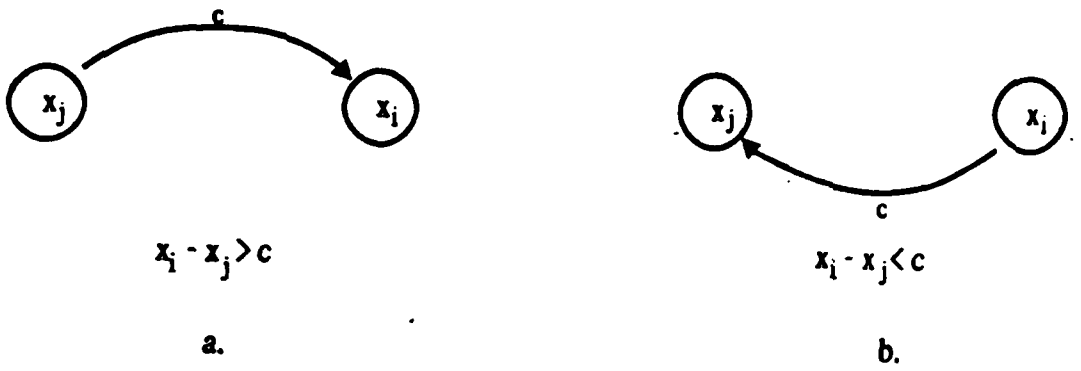


Figure 6.4. Upper and Lower Bound Constraints

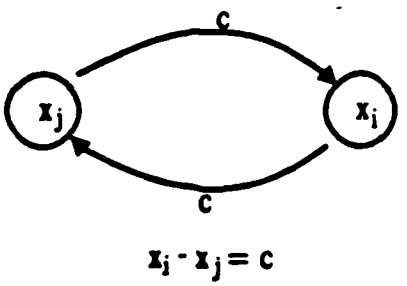


Figure 6.5. Equality Constraints

introducing an arc pointing from the successor node to the predecessor node in their ordering. The graph representation of (6.1) is shown in Figure 6.4a, and that of (6.2) is shown in Figure 6.4b. The equality constraints are represented by using the lower-bound and upper-bound constraints simultaneously. The graph representation of (6.3) is shown in Figure 6.5. The arc from node x_j to x_i is a regular arc which represents a lower-bound type constraint. The arc from node x_i to x_j is an arc whose direction is reversed. Let us term these arcs as left-directed arc and bottom-directed arc in the horizontal graph and the vertical graph, respectively. They create a non-positive cycle in the graphs G'_x. Their algorithm repeats the acyclic graph algorithm up to a number of left-directed (or bottom-directed) arcs. This algorithm requires iteration of the acyclic algorithm but, it is more efficient than the most general Ford-Bellman algorithm. The algorithm detects the inconsistent constraints. In order to handle the equality and upper-bound type constraints, their algorithm can be used for constructing the longest path spanning tree in our compaction method.

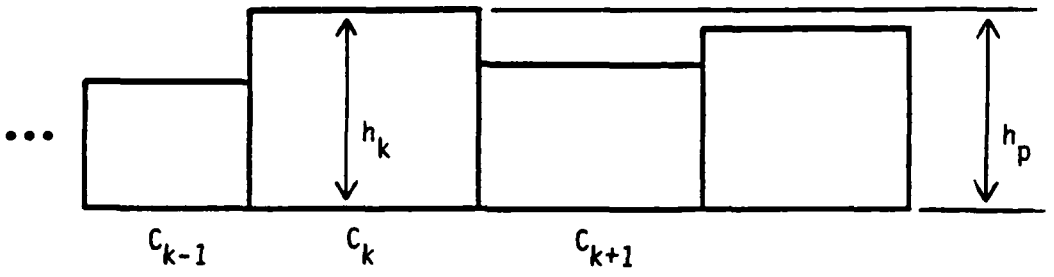


Figure 6.6. Height of Data Path

6.2.2. Minimum Objective Dimension

In designing a VLSI circuit, the whole layout is subdivided into blocks, which are designed independently and placed together. In this circumstance, it is quite likely that certain blocks do not need to be compacted less than some height or width. Further compaction more than a given goal only increases dead area between blocks. Let us call this kind of goal *minimum objective dimension*. If a minimum objective dimension is given only in one direction, the algorithm should take advantage of resulting loose space to compact tighter in the other direction. We describe the design method using a data path as an example.

Design of an integrated circuit as a data path is an approach of the structural design method described by Mead and Conway [MeC80]. The OM-2 chip is an example of a chip designed using this approach. One possible approach for designing a data path is as follows. A single bit data path is sub-divided into sections and each section can be designed separately as a basic cell. Then, they can be placed side by side to construct a single bit data path. Finally, it is duplicated to the width of the data path. Let us assume there are n basic cells for a data path. We denote them as $C_1, ..., C_n$. We can design and compact them independently. Assume that the data path runs in the horizontal direction, and let the height of each cell be represented by h_i for $i = 1, ..., n$. Let h_p be defined as:

$$h_p = \max_i \{h_i\} .$$

Then h_p is the height of the single bit data path, assuming we do not need an extra expansion related to connection points. Assume that $h_p = h_k$ as shown in Figure 6.6. Then we should recompact all the cells but cell C_k with the minimum objective height of h_p. All we need to incorporate this goal into our algorithm is the introduction of a simple constraint arc. The arc points from the source node y_s to the sink node y_t and its weight is h_p. Figure 6.7 represents cell C_i, where $i \neq k$, along with cell C_k and a simple arc introduced in the vertical graph of cell C_i.

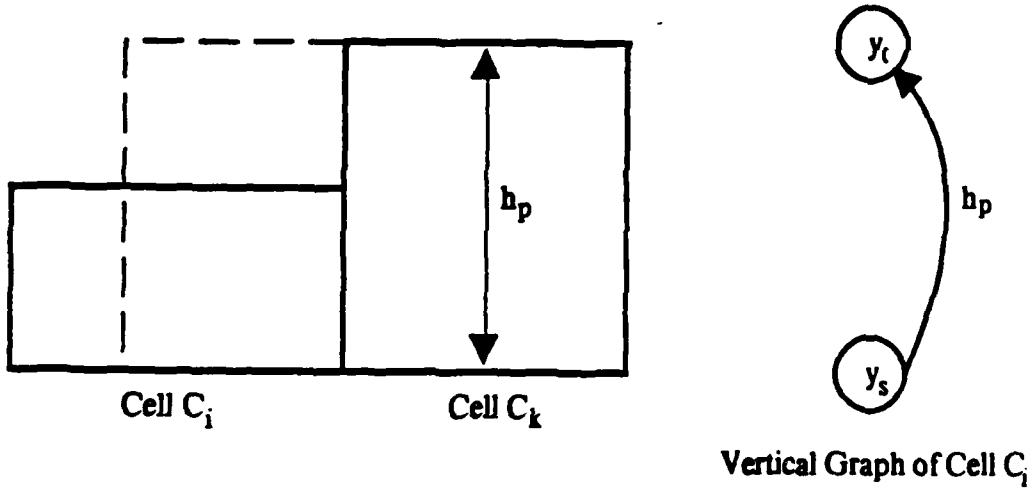


Figure 6.7. Minimum Objective Height

No modification of any algorithm is necessary to take advantage of this goal and to attempt to compact more tightly in the horizontal direction. A goal of minimum required width is incorporated similarly.

Simple examples of the minimum required dimension are shown in Figure 6.8 and Figure 6.9. They are two layouts for an inverting supper buffer. The layout of Figure 6.8 is produced with no minimum objective dimension. The layout of Figure 6.9 is compacted with the minimum objective height of 51λ. Their stick diagram inputs are identical, but the graph corresponding to the layout of Figure 6.9 has one extra arc from its source node to sink node. As a result of the minimum objective height, the width of the latter layout is narrower than the former. This results from the two dimensionality of our compaction algorithm. The dimensions of the two layouts are shown in Table 6.1.

	Height(λ)	Width(λ)	Area(λ^2)
Layout of Figure 6.8	38	22	836
Layout of Figure 6.9	51	18	918

Table 6.1. Dimensions of Super-buffers

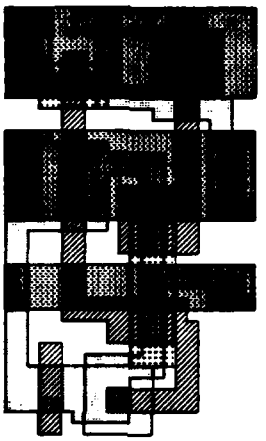


Figure 6.8. Super-buffer

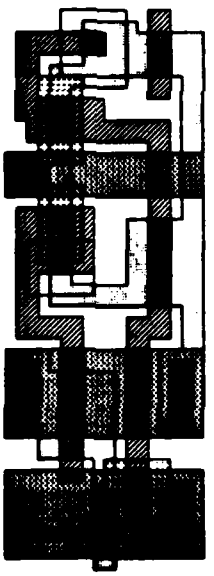


Figure 6.9. Super-buffer Fixed Height

6.2.3. Preferred Direction of Compaction

Another method to influence compaction is to give a preferred direction of compaction. In the previous example of the data path, it may be reasonable to attempt to recompact the highest cell C_k further in the vertical direction, accepting a larger horizontal dimension. It can be done by using the following objective function:

$$f(x,y) = t_x \cdot t_y + K \cdot t_y$$

where K is some constant which represents a degree of preference. If K is larger, there is a greater tendency to compact in the vertical direction. The slight adjustment in the formula for selecting the next branching arc is also necessary. The detail of the algorithm is described in section 5.1.1. Since the objective function is not simply an area of layout but an area plus weight that represents preference, the value of the objective function, pseudo-area, f is now defined as

$$f = t_x^u \cdot t_y^u + K \cdot t_y^u .$$

The estimate of the pseudo-area after branching is now defined as

$$\begin{aligned} f' &= (t_x^u + \Delta x)(t_y^u + \Delta y) + K \cdot (t_y^u + \Delta y) \\ &= (t_x^u \cdot t_y^u + K \cdot t_y^u) + t_x^u \cdot \Delta y + t_y^u \cdot \Delta x + \Delta x \cdot \Delta y + K \cdot \Delta y \\ &= f + \Delta f . \end{aligned}$$

Therefore, the function Δf used for selection of the next branching arc is

$$\Delta f = t_x^u \cdot \Delta y + t_y^u \cdot \Delta x + \Delta x \cdot \Delta y + K \cdot \Delta y .$$

Other parts of the algorithm do not need to be modified. It may, however, be helpful to treat vertical compaction favorably in the heuristic algorithm for an initial guess. The preference to the horizontal direction is similarly incorporated.

6.3. Graph Modification for Recompaction

In this section, modification of the graph is considered as a further compaction method. It is a costly operation because it involves the iteration of the compaction process described in the previous chapter. The original graph is modified using information from the current compacted layout in an attempt to achieve additional compaction. Then the compaction process is repeated. We can use the result of the previous compaction as a basis for an initial solution of the next compaction. The modification of the graph can be used for automatic jog point insertion. It also can be used to overcome the restriction of our formulation, that is, the limitation to the two way choices.

6.3.1. Jog Point Insertion

Under certain conditions, insertion of a jog point in a wire segment is necessary to compact a layout further. An algorithm that detects a place to insert a jog point in a wire segment is described by Hsueh [Hsu79]. In his representation, a group is represented by a single node. However, his algorithm of detection also works in our representation. There are two conditions that should be distinguished for jog point insertion in our representation. Under one condition, the algorithm works more straightforwardly. In the other, it works exactly as described by Hsueh for his Cabbage system. Let us discuss the algorithm using an example of the longest path of the horizontal direction.

The first condition is the case that the longest path from the sink to the source enters a vertical group in one node and leaves it from another node as shown in Figure 6.10. Since a group consists of alternating symbols and wires, one of the nodes along the longest path in this group represents a wire. The corresponding layout for Figure 6.10 is depicted in Figure 6.11. The wire along the longest path is possibly a place in which a jog point may be inserted.

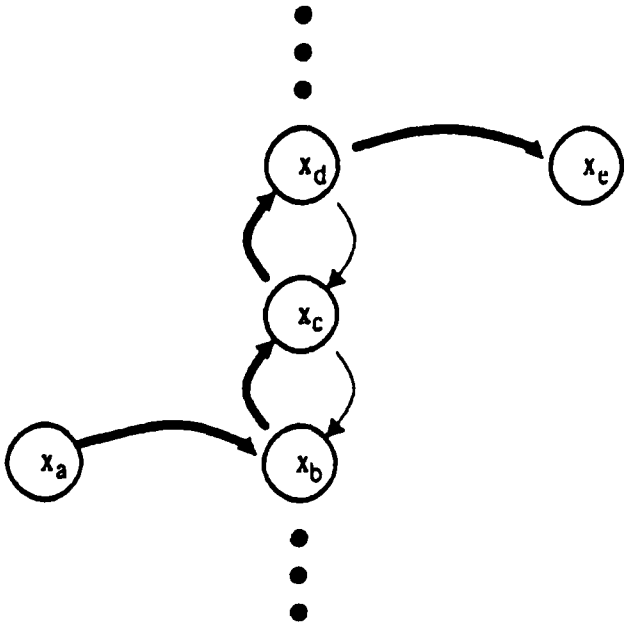


Figure 6.10. Longest Path Along Vertical Group

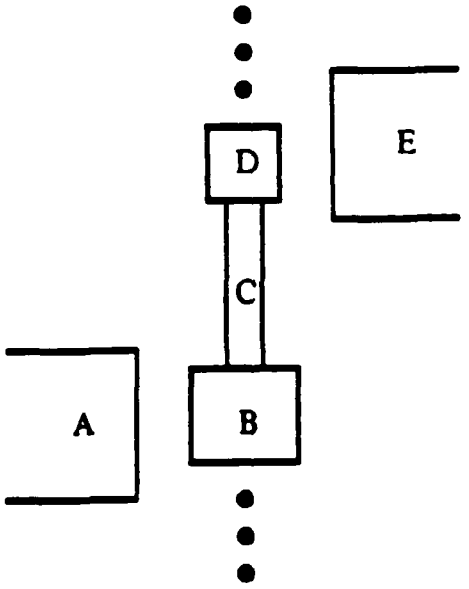


Figure 6.11. Layout Example

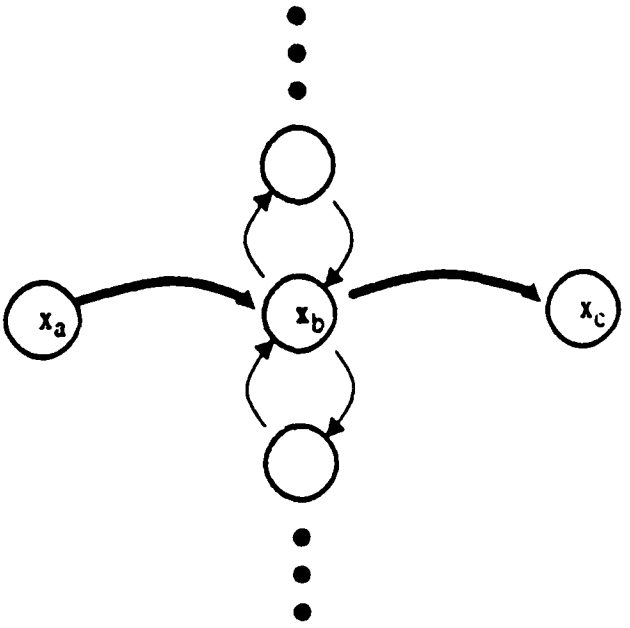


Figure 6.12. Longest Path

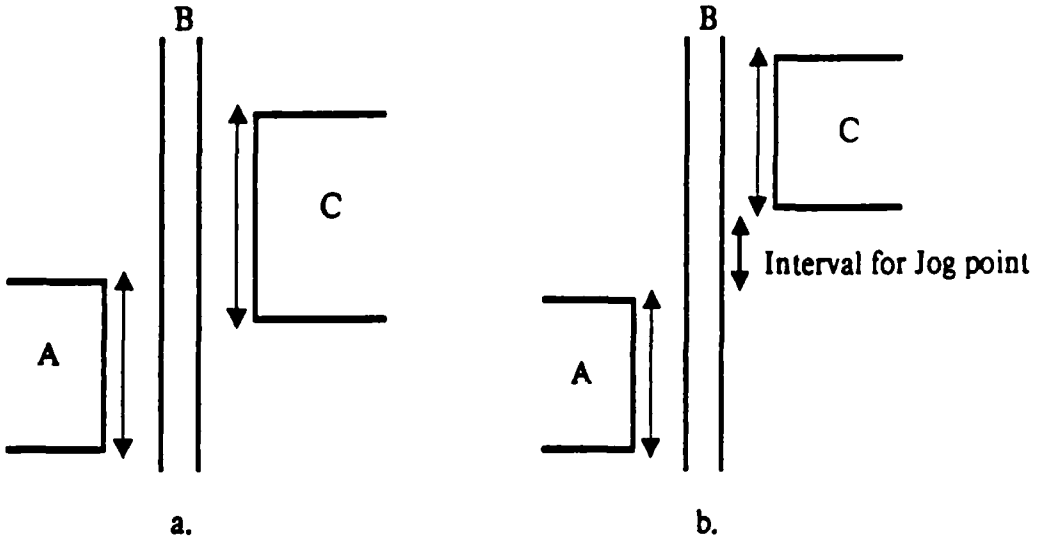
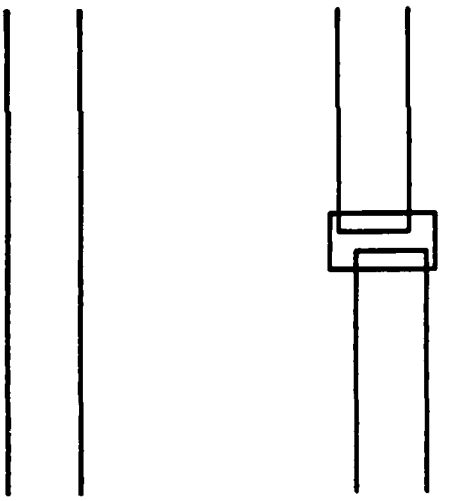


Figure 6.13 Layout Examples

The second condition is that the longest path enter a vertical group in one node and leave it from the same node. If the node represents a symbolic element, there is no place where one can enter a jog point in this vertical group. However, if the node represents a wire segment, it may be advantageous to enter a jog point in that wire. The graph representing this condition is shown in Figure 6.12 in which nodes are named as x_a, x_b and x_c, and node x_b represents the x-coordinate of wire B. This is the case in which we need a further check similar to the one described by Hsueh. It is necessary to distinguish an interval of wire B that is directly constrained by existence of the element A, and a similar interval related to the element C. If these two intervals do not overlap, then wire B is a possible location for a jog point insertion. These intervals can be computed from current y-coordinates of involved symbols, their vertical sizes, and the design rules. Figure 6.13 represents layouts corresponding to the graph of Figure 6.12. Figure 6.13a shows the layout in which two intervals overlap and wire B is not a candidate for jog point insertion. Figure 6.13b shows the situation where the two intervals do not overlap.

From all possible wires for jog point insertion, one should choose the wire with the longest interval for jog insertion. In the first case, the interval for jog insertion is the length of the entire wire. In the second case, the interval for jog insertion is the distance between the two constraining intervals, because the jog point must be inserted between these intervals.



a. Straight Line b. Jogged Line

Figure 6.14. Jog Point

The jog point insertion is done by separating a vertical wire into two wires and inserting a single horizontal wire as shown in Figure 6.14. This separates a single vertical group into two separate vertical groups and an entirely new horizontal group is generated between them. The two vertical groups just separated are ordered appropriately to take advantage of the jog point. In the example of Figure 6.13b, the upper group is defined as the predecessor of the lower group. According to the graph generation algorithm described in Chapter 4, new arcs and related decision variables have to be generated and added to the already existing ones. Also,

one simple arc that originally connected two terminating elements of the bent wire, that is, the separated vertical wire, should be eliminated. It is not necessary to change the current setting of existing vector δ. One executes the heuristic algorithm for initial estimation for newly introduced 0-1 variables with all the old δ's being fixed to the result of the previous compaction. Then, the compaction process proceeds exactly as described in Chapter 5.

6.3.2. Three Way Choices

Our graph theoretic formulation has the limitation that it can express only the two way choices as described in Section 3.5.4. One possible method to cope with this restriction is to use a modification of the graph and recompaction. In order to discuss the method, assume the following input and output layouts. The input stick diagram is depicted in Figure 6.15. The topological relation in the result of compaction is shown in Figure 6.16. Under this condition, it may be possible to compact further by modifying the current graphs and performing the compaction operation.

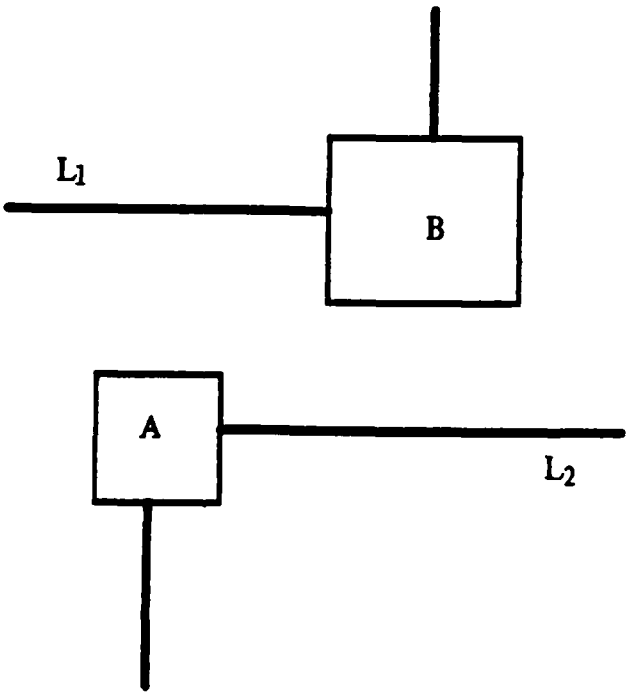


Figure 6.15. Relative Position in Input Stick Diagram

The method to detect a place for the graph modification is to find a pair of groups whose final relative positioning is the reverse of their group ordering. If graphs are generated from the layout of Figure 6.16, the ordering of the two groups to which symbols A and B belong is exchanged. Therefore, in the modification of the graph, all dual arcs between elements in these two groups are affected; the direction of the horizontal components of those dual arcs is reversed. The vertical components of the dual arcs are not changed. The change in ordering also affects a choice of simple arc and dual arcs between wires and other elements. This happens if the wires' terminating symbols belong to the group whose ordering has been changed.

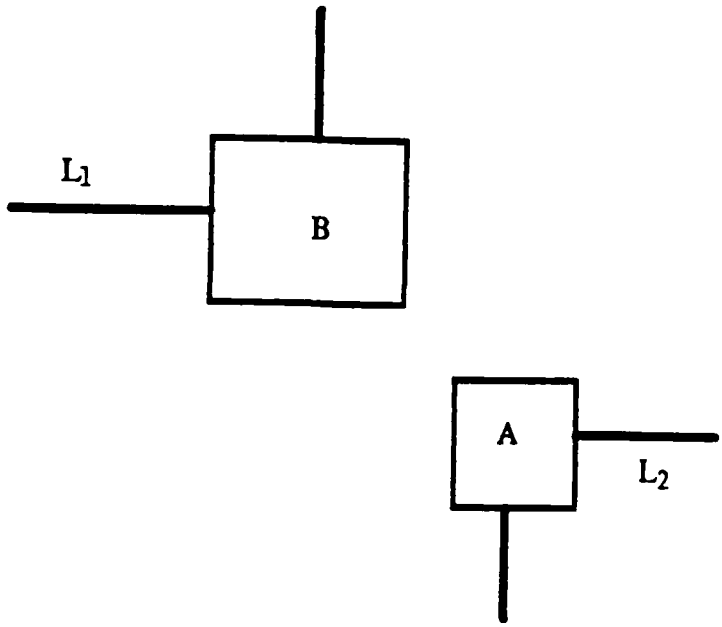


Figure 6.16. Relative Position after Compaction

For example, there are simple arcs between wire L_1 and symbol A and between wire L_2 and symbol B in the original layout of Figure 6.15. However, dual arcs replace these simple arcs in the graph generated from the layout of Figure 6.16. These changes in choice of a simple arc and dual arcs can be derived directly from the rules described in Section 4.4.3.

The current value of vector δ can be used as the initial estimate of the succeeding compaction process. The current compacted layout is the initial estimate of the optimal layout to be pursued.

6.4. Block Level Placement

In order to cope with the complexity of VLSI, hierarchical layout methods have been investigated [Pre79, PrG79, SzO80]. The original VLSI system is sub-divided to sub-blocks and each sub-block is laid out individually. Then, placement and routing algorithms are used to complete the whole layout [CiK82, Prv79, Pre79]. The graph-based formulation presented here can be used for the placement algorithm if the initial rough placement is given [CiK83]. The initial rough placement can be obtained using a mini-cut algorithm [Lau79]. The data used by Ciesielski [CiK83] is shown in Figure 6.17. This data has been compacted by our method and the result is shown in Figure 6.18.

6.5. Minimization of Wire Length

If the longest path method is used for compaction, elements tend to move in the direction of the source node as much as possible. By the longest path method, the coordinates of elements on the longest path are fixed. However, for elements not on the longest path there are slack spaces associated with them as discussed in Section 5.7.2. The longest path algorithm

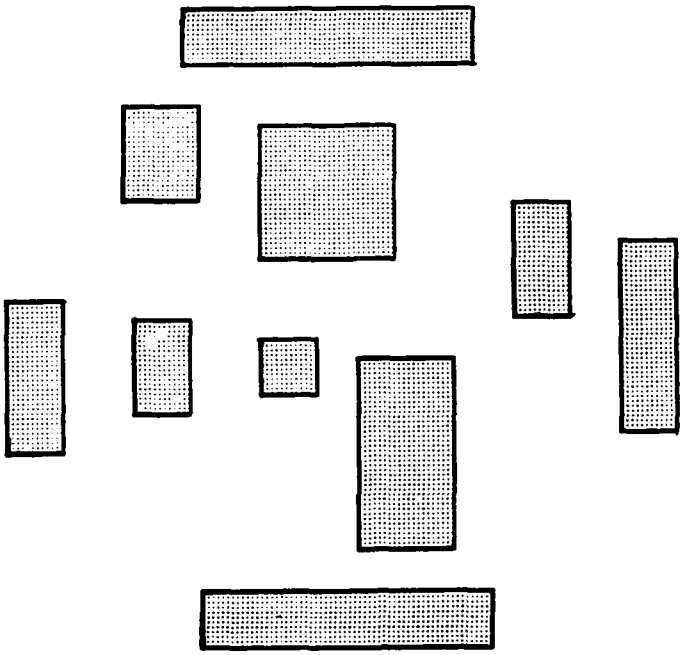


Figure 6.17. Input for Block Placement

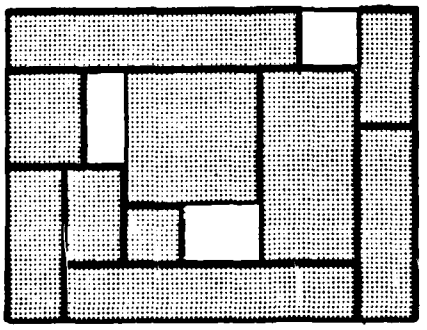


Figure 6.18. Result of Placement

usually assigns these elements the least coordinate. This may cause needlessly long wiring. The example is shown in Figure 6.19. It represents a part of the layout for a bit slice ALU. At the left side of this layout, two diffusion wires and one polysilicon wire are unnecessarily

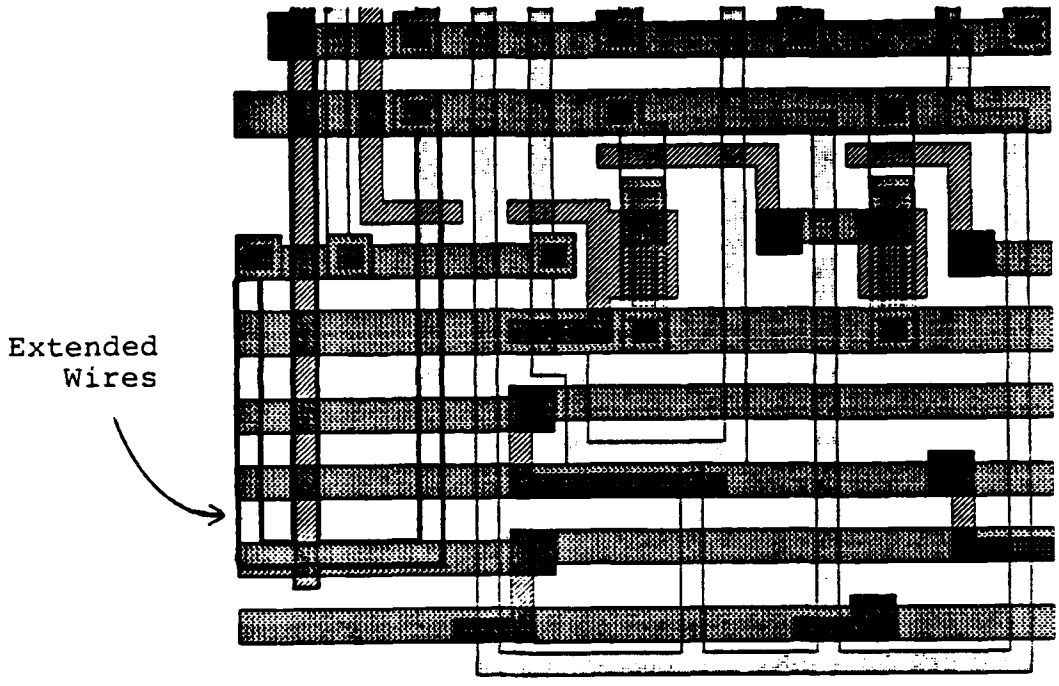


Figure 6.19. Extended Wires

extended, because the horizontal group consisting of two diffusion wires and an enhancement transistor is placed toward the bottom of the layout. This problem was discussed by Hsueh [Hsu79]. He used a predecessor-to-descendent ratio of each group to distribute the slack space to two sides of the group.

We can formulate this problem more formally and can solve it to minimize the wire length. After the compaction process is terminated, the position of the sink nodes can be finalized. This insures all of the coordinates of the elements on the longest path to be fixed. The size of the layout is finalized by the following two equations; for horizontal direction:

$$t_x = w \tag{6.1}$$

for vertical direction:

$$t_y = h \tag{6.2}$$

where w and h are the height and width of the layout resulting from the compaction process. After finalizing the dimensions of the layout, it is possible to formulate two independent linear programming problems to minimize the total wire length. Remember that each arc in our formulation corresponds to an inequality imposed by design rules and connection requirements of the design. The constraints of the linear programming problems are the inequalities represented by the final graphs plus the equality (6.1) and (6.2). For horizontal direction, the constraints are the inequality represented by the arcs in

$$A_x \bigcup R_x \bigcup D_x^u(\delta^n)$$

and equality (6.1) where δ^n is the final setting of 0-1 variables. The objective function is:

$$C_d \sum_{i \in W_d} |x_i' - x_i| + C_p \sum_{i \in W_p} |x_i' - x_i| + C_m \sum_{i \in W_m} |x_i' - x_i| \tag{6.3}$$

where W_d is a set of diffusion wires in the layout, W_p is a set of polysilicon wires in the layout, and W_m is a set of metal wires in the layout. For the i-th wire, x_i and x_i' is the coordinate of terminating points. C_d, C_p, and C_m are arbitrary constants. The above objective function is not linear, but it can be transformed to a linear function by variable substitution. Let us introduce two variables, x_i^+ and x_i^-, for each wire $i \in W_d \bigcup W_p \bigcup W_m$. They are defined as

$$\begin{aligned} x_i^+ &\geq x_i - x_i' \\ x_i^+ &\geq 0 \end{aligned} \tag{6.4}$$

$$\begin{aligned} x_i^- &\geq x_i' - x_i \\ x_i^- &\geq 0. \end{aligned} \tag{6.5}$$

Now the objective function (6.3) can be linearized as the following:

$$C_d \sum_{i \in W_d} (x_i^+ + x_i^-) + C_p \sum_{i \in W_p} (x_i^+ + x_i^-) + C_m \sum_{i \in W_m} (x_i^+ + x_i^-).$$

The constraints of the problem must be supplemented with (6.4) and (6.5). For the constant coefficients, one can set

$$C_d = C_p = C_m = 1$$

for the simplest case. Here, all the wires are equally minimized so that the total length of the wires is minimized regardless of their layer. However, total wire length can be controlled depending on layers. For example, one can have

$$C_d > C_p > C_m > 0.$$

In this case, diffusion wires have the highest priority to be shortened followed by polysilicon wires, and metal wires have the lowest priority. Therefore, the diffusion wires are most rigorously minimized, but the longer length of metal wires may be tolerated.

6.6. Soft Cells

In automatic layout systems, it is common to have basic units of integrated circuits stored as library cells [PDS77 Row80b]. With an efficient compaction procedure, it is very effective to have library cells in the format of a stick diagram that is stored as constraint graphs. A compaction algorithm can construct cell layouts from stick library cells as necessary. It can take into account individual conditions such as user (or system) supplied goals. As a result, the library cells behave as soft cells. They can be compacted more tightly in one direction or the other to take advantage of an individual situation, and they can be stretched to satisfy connection and other requirements. The ability of the compaction algorithm to incorporate additional goals is important for using soft library cells.

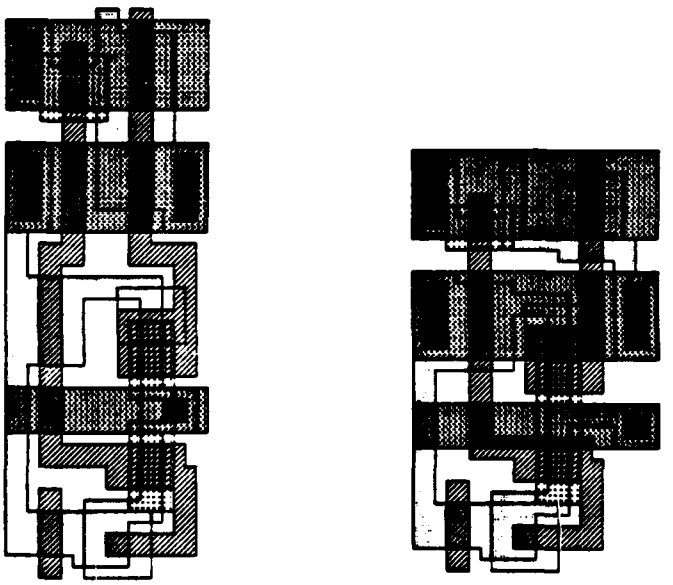
In the following sections, two more cases where soft cells are advantageous are presented.

6.6.1. Automatic Rescaling of Ratio and Driving Power

NMOS logic gate performance depends on the width/length ratio of component transistors. Therefore, they are called ratio logic. According to the Mead and Conway style of design methodology[MeC80], a minimum inverter has a pull-up and pull-down ratio of four. However, if input to an inverter is supplied through a pass transistor, a designer is required to make a pull-up to pull-down ratio of eight instead of four. This is necessary to maintain the same output voltage level in spite of lower input voltage. The input supplied through pass transistors is one threshold voltage lower than the regular voltage that represents the high state.

If library symbols are defined with stick diagrams, it is easy to generate cells of the same logic with different transistor ratios. The topological connection among symbols and wires are the same but the sizes of transistors are slightly different. In terms of the graphs, their structure is same, but weights of some arcs are changed according to the size of transistors. There is a procedure that traces a network of transistors in a layout and calculates a voltage level of each node [BaT80]. Therefore, one can construct a system which automatically rescales a transistor ratio and lays out a compacted cell. For this purpose, a single topological representation, therefore a single graph representation, is needed as a library cell.

Similarly, transistors' sizes can be changed according to the load driven by them. Examples of super buffer layouts are given in Figure 6.20. The super buffer of Figure 6.20a has twice the driving power of that of Figure 6.20b. They are produced from the same topological definition of the library cell; the only modification is the rescaling of the size of two transistors. The corresponding inputs for the layouts in Figures 6.20 are shown in Figure 6.21. Our algorithm is useful as a part of a silicon compiler that can rescale transistors according to a



a. b.

Figure 6.20. Super-Buffers

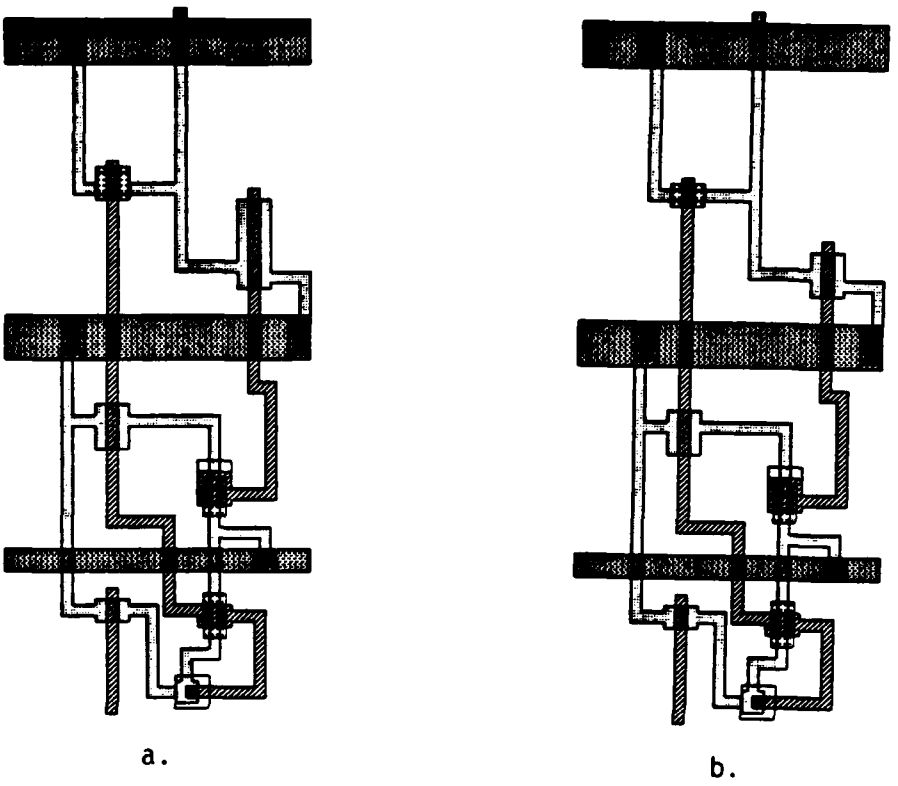


Figure 6.21. Input Stick Diagrams for Super-Buffers

driving load and to input voltage level.

The rescaling of transistors can also be used to create more compact layout. One can change the size of pull-up transistors and pull-down transistors in concert, and maintain a ratio of an inverter or other gates. In certain conditions, one set of transistors may create smaller layouts than others. Power consumption is also changed if one uses transistors of different size. There is an area and power consumption trade off. In certain cases, one should lay out a cell from a stick library cell to minimize power consumption.

6.6.2. Robustness against Technological Change

The flexibility of a soft cell represented by a stick diagram gives an advantage when design rules are changed. The design rules can be changed when a fabrication process is changed or a minimum feature size is reduced. Since the fabrication technology is progressing rapidly, it is essential to devise a method that prevents previously designed cells to become obsolete. If the cell is tightly laid out, it is non-trivial work to make any change according to

new design rules. If the cell is represented in a stick diagram, changes can be done much more easily.

For example, MOSIS[1] originally supported a butting contact. Their design rules are almost same as the ones described in Mead and Conway [MeC80]. However, as feature size becomes smaller, butting contacts force the metal layer to connect to diffusion and polysilicon layers on fairly steep surfaces. Consequently, they do not guarantee a high yield, if butting contacts are used. Instead of butting contacts, buried contacts should be used. Cells laid out with butting contacts must be redesigned with buried contacts. Note that all the pull-up transistors must also be redesigned, because they use butting contacts. If the cells are tightly laid out, re-design is not an easy task. The space requirements around butting contacts and buried contacts are different. Also, the pull-up transistors with buried contacts are larger than the ones with butting contacts. Re-designing is much easier at the stick diagram level. Figure 6.22 shows a loosely drawn, stick diagram equivalent, layout of a bit slice ALU cell [DHS82]. The cell is originally drawn using butting contacts. The result of compaction is depicted in Figure 6.23. In order to agree with MOSIS recommendation, the cell is redesigned using buried contacts. No stand along buried contact is used in the design. The pull-up transistors with butting contacts are replaced the pull-up transistors with buried contacts. Also, at two places, connections are made to the metal layer of the pull-ups taking advantage of the existence of a metal layer in the butting contacts. A metal diffusion contact and a metal polysilicon contact are introduced as replacements. The result of re-design and re-compaction is shown in Figure 6.24. This is larger than the original, but the re-design process requires a minimum amount of operation. Small re-wiring at a stick diagram level is done, which is aimed to take advantage of the fact that buried contacts do not include a metal layer. The result of this re-wiring and compaction is shown in Figure 6.25. The result is even smaller than the original layout with butting contacts. Size and area of the three layouts are shown in Table 6.2.

	Height(λ)	Width(λ)	Area(λ^2)
Layout of Figure 6.23	101	72	7272
Layout of Figure 6.24	106	75	7950
Layout of Figure 6.25	98	73	7154

Table 6.2. Dimensions of ALU's

[1] Silicon foundry service supported by Defence Advanced Research Project Agency and operated by Information Science Institute of University of Southern California.

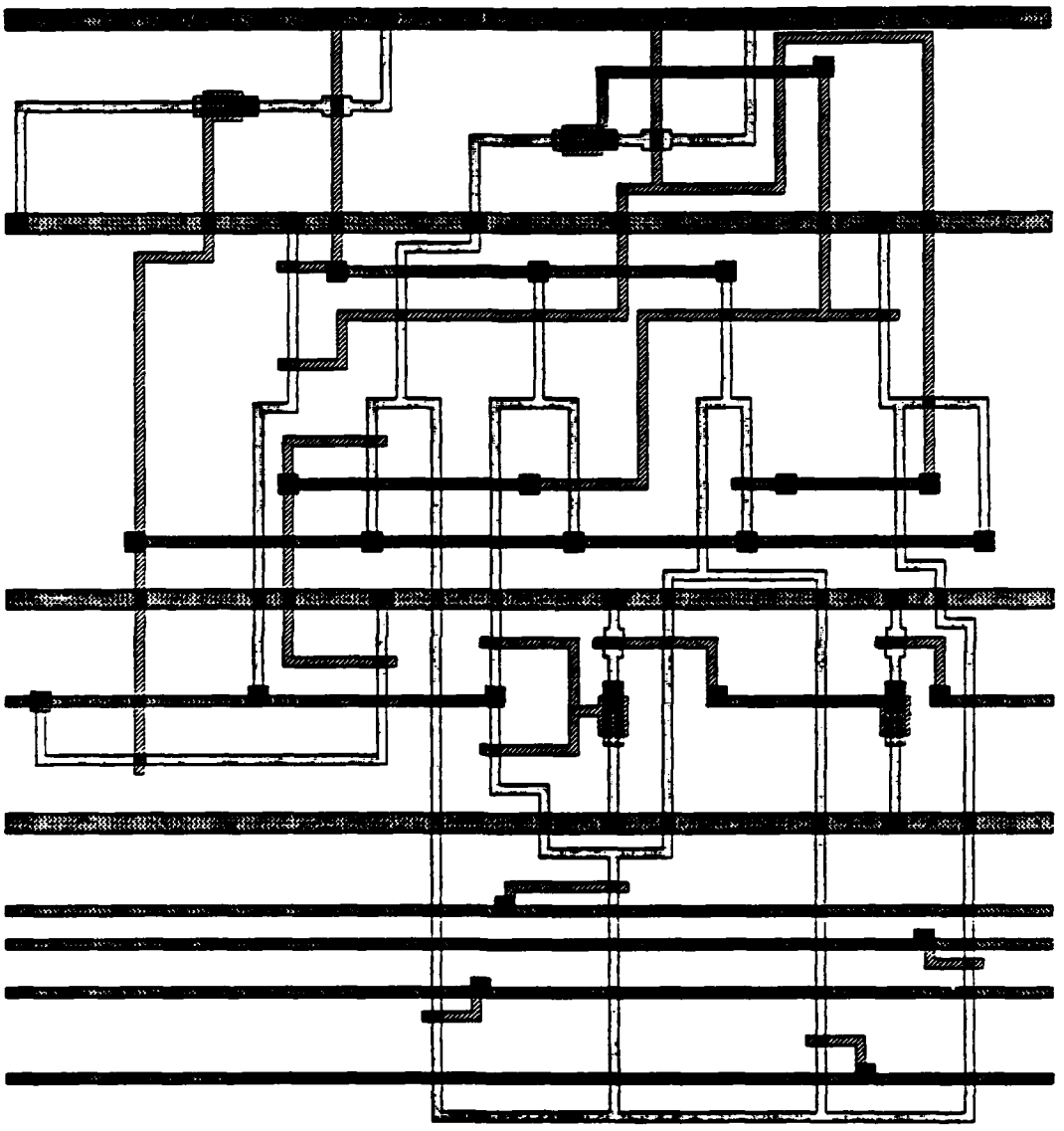


Figure 6.22. Input Stick Diagram for ALU

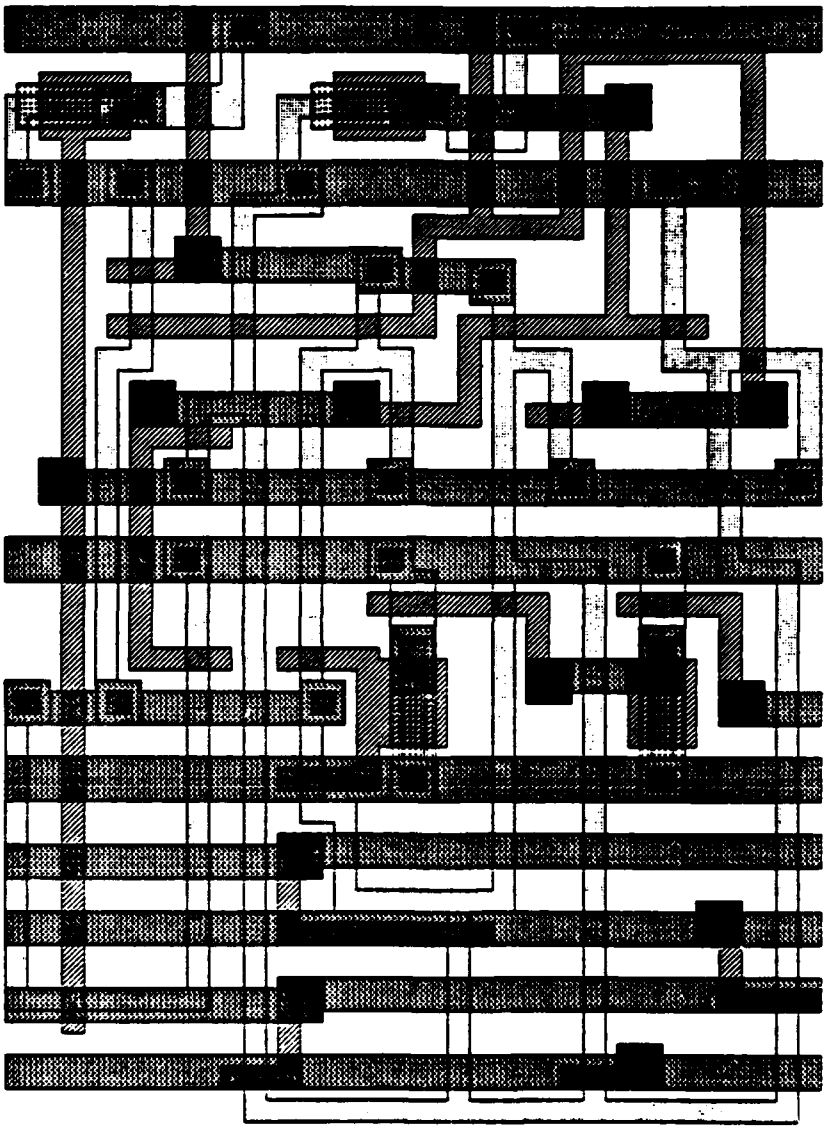


Figure 6.23. ALU with Butting Contacts

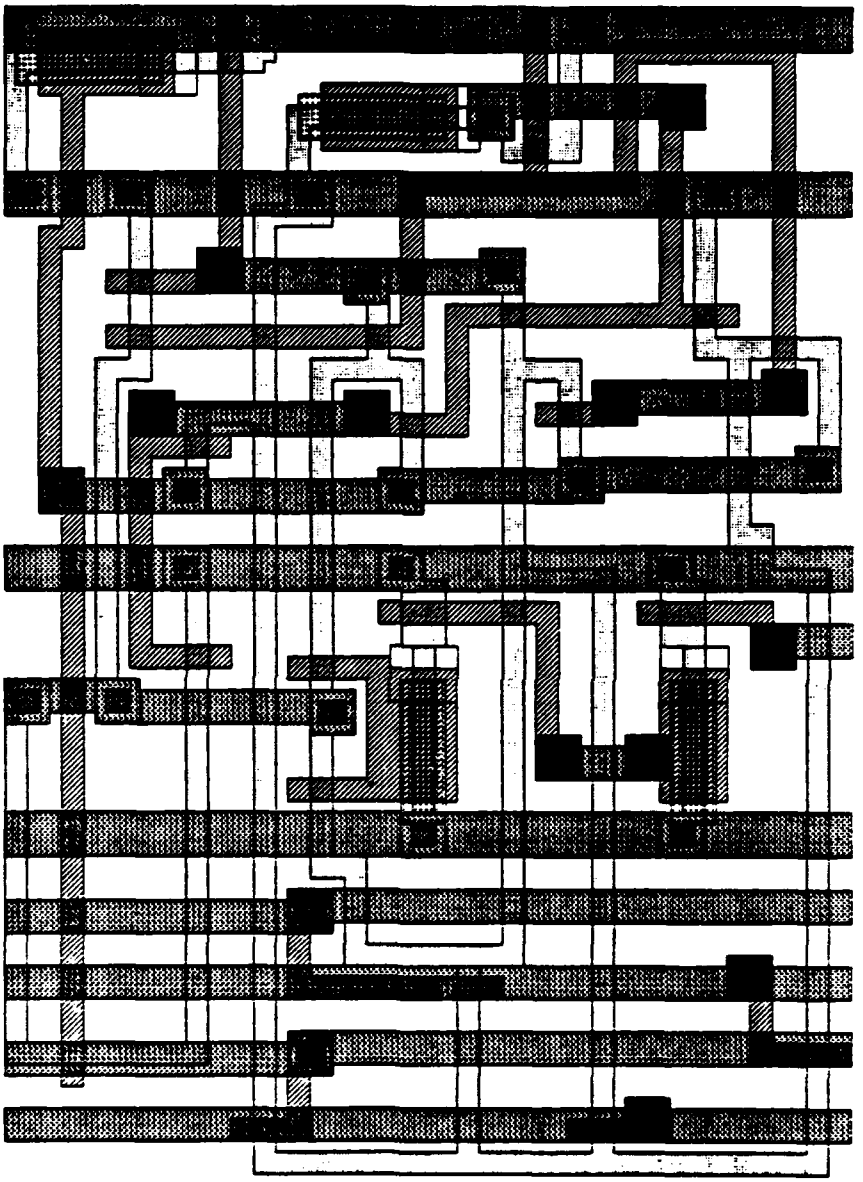


Figure 6.24. ALU with Buried Contacts

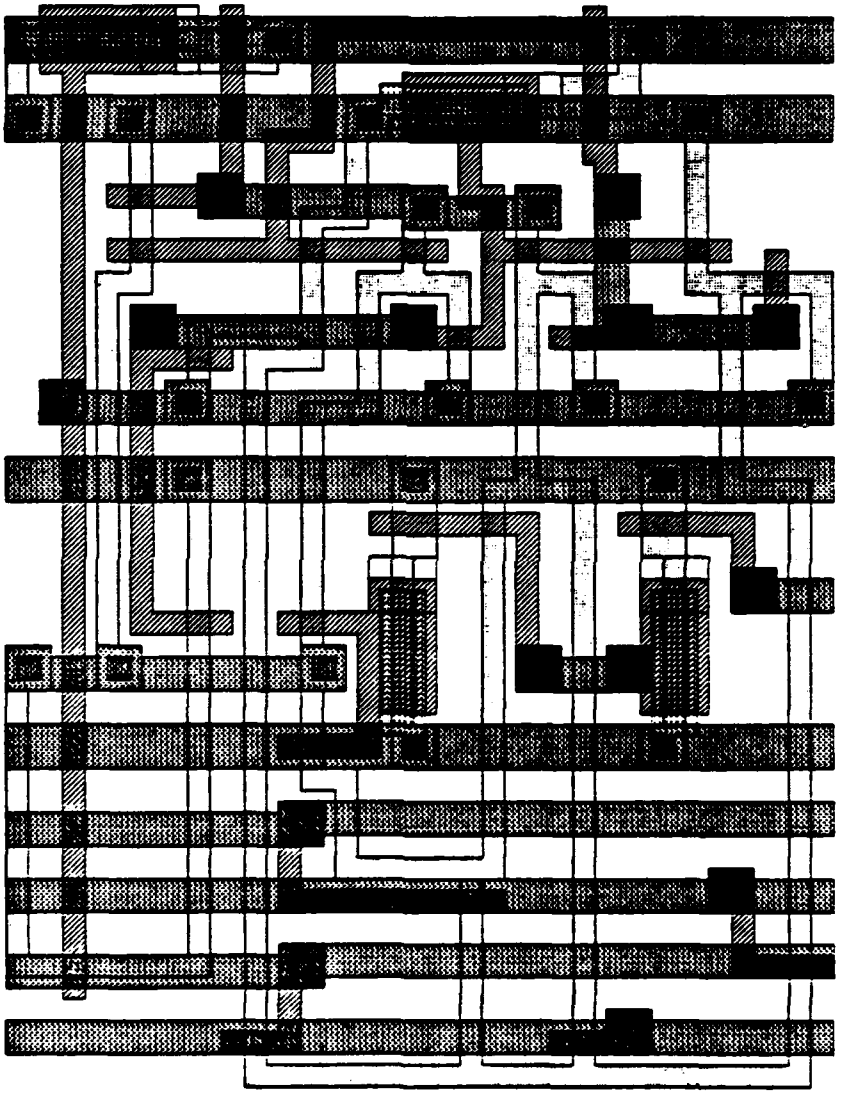


Figure 6.25. Redesigned ALU with Buried Contacts

CHAPTER 7

Experimental Implementation and Results

7.1. Experimental Implementation

The experimental program, **squash,** was constructed to test our formulation and algorithms. The program was written as a research tool. Consequently, certain decisions were made to facilitate the implementation sacrificing fancy user interface. The program was written in PASCAL. It was developed on a VAX-11/780 running the UNIX[1] operating system. CIF is used both as input and output languages. The input language is a subset of CIF, and a number of restrictions are made to remove the burden from the input parsing and graph construction process of the program. The rules for preparing input for the program are described in Appendix A. The library symbols are predefined. This decision was made to facilitate the implementation. User defined library symbols can be implemented without any change in the compaction part of the program. The predefined symbols are shown in Figure 7.1.

The program consists of two major parts: preprocessing and compaction. The preprocessing part of the program reads in CIF input and constructs the graph structure used for compaction. The program parses CIF input by a recursive descent algorithm. Since the CIF language only represents geometrical information, the program has to extract all other information. The program extracts electrical nodes and then constructs a network of electrical nodes. It then performs the scan line based algorithm to construct the compartment information as described in section 4.5.1. Finally, it constructs the graph structures that are used by the compaction part of the program. The algorithms used in the preprocessing part are described in Chapter 4.

The compaction part of the program consists of two parts: an initial estimation procedure and a branch and bound procedure. The algorithm used for initial estimation is explained in section 5.6. The branch and bound method is described in Chapter 5. In the current implementation, depth first search method is used in the branch and bound algorithm. The depth first method is useful in order to find the optimal or a sub-optimal solution quickly. It is also fairly straightforward to program using a recursive procedure. Good termination conditions are not yet formulated.

7.2. Results

Several integrated circuit layouts have been constructed using the program as experiments. The program has also been released to a limited user community.

Typical examples of compaction are presented. They are ordered according to the number of the elements, that is, symbols and wires, in the layout. The number of elements does not necessarily represent the complexity of the problem. Table 7.1 shows the size and the complexity of the input examples, and contains the following information:

(1) element -- the number of elements (i.e., symbols and wires).

[1] UNIX is a Trademark of AT&T Bell Laboratories

Figure 7.1. Library Symbols in Squash

(2) x-node -- the number of nodes in the horizontal graph.

(3) x-arc -- the number of arcs in the horizontal graph.

(4) y-node -- the number of nodes in the vertical graph.

(5) y-arc -- the number of arcs in the vertical graph.

(6) 0-1 var -- the number of 0-1 variables (the number of pairs of dual arcs).

Table 7.2 shows the results of the program execution using the examples of Table 7.1. It contains the following information:

(1) init area -- the area computed by the heuristic initial estimation procedure.

(2) best area -- the best area computed by the compaction process.

(3) improved -- percentage of improvement made by the branch and bound on the initial estimation.

(4) init iter -- the number of iterations performed by the heuristic initial estimation procedure.

(5) update -- the number of times that the branch and bound routine updates the best solution.

(6) best BB -- the branching that finds the final best solution.

(7) optimum -- the branching that verifies the optimality of the solution. Question mark (?) indicates that the optimality is not verified by the compaction process.

	scell	sbuf	tff	prcell	alu	minmax	crc	edge
element	26	57	69	210	221	269	282	394
x-node	15	31	47	135	137	182	168	253
x-arc	44	93	156	492	544	838	956	1050
y-node	17	37	43	133	144	159	178	265
y-arc	47	138	113	476	656	599	686	1122
0-1 var	14	187	39	1052	1896	3005	4406	2445

Table 7.1. Input Examples

	scell	sbuf	tff	prcell	alu	minmax	crc	edge
init area	432	988	2067	8307	8320	9948	10680	32512
best area	432	836	2014	7381	7272	7467	7620	30988
improved	0%	15%	3%	11%	13%	25%	29%	5%
init iter	5	27	12	61	56	56	104	45
update	0	3	1	9	12	21	23	6
best BB	0	14	1	89	48	84	154	19
optimum	1	?	12	?	?	?	?	19
final BB	1	300	12	300	300	300	300	19
bound	-	0%	-	2.5%	2%	6.2%	13.2%	-

Table 7.2. Result of the Compaction

(8) final BB -- the last branching.

(9) bound -- the worst case bound of the compaction result from the optimal.

Table 7.3 shows execution time of the above computation. It displays the following information:

(1) preproc -- computing time in cpu seconds to the end of preprocessing.

(2) init -- computing time to the end of the heuristic initial estimation process.

(3) best BB -- computing time to the end of the branching that finds the best solution.

(4) final BB+ -- computing time to the end of the branch and bound including the compacted layout output.

The input stick diagrams and the compacted layouts for all the examples are shown in Appendix B.

7.3. Discussion

The first example, scell, is a layout of a shift register cell [MeC80]. This example is the smallest. The initial estimation computed by the heuristic method is the optimal solution. The optimality is verified in the first branching and the compaction process terminates.

The second example, sbuf, is a layout for a super buffer cell [HoS80]. The third example, tff, is a layout for a t-flip-flop [Hsu79]. The number of elements in the t-flip-flop is larger than the number of elements in the super buffer cell. The super buffer cell produces more 0-1 variables than the t-flip-flop. The optimality is verified for tff but not for sbuf. Not only the size of the layout but also the structure of layout affects the size of the resulting compaction problem.

The fourth example, prcell, is a hardware implementation of a priority queue [Ked81]. This cell was originally laid out by hand. The result of compaction is 3% smaller than the one designed by hand. The fifth example, alu, is a simple bit slice ALU [DHS82]. The sixth example, minmax, is a serial comparator. It accepts as input two integers serially with the most significant bit first and outputs the larger and the smaller numbers. This cell was also originally designed by hand. The cell generated by **squash** is 2% larger than the hand designed one.

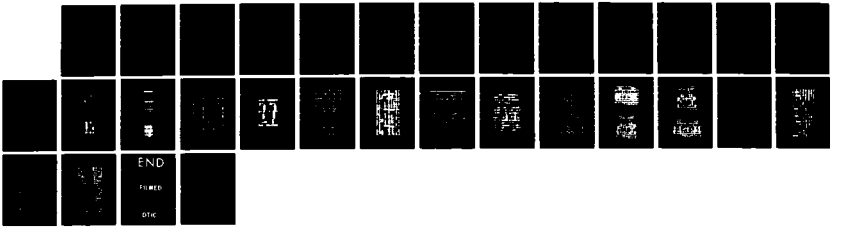
The seventh example, crc, is a cell which computes a CRC bit from a serial input. This cell is not much larger than prcell, alu, or minmax. However, its design includes a greater

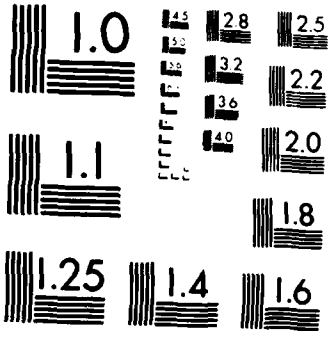
	scell	sbuf	tff	prcell	alu	minmax	crc	edge
cumulative time								
preproc	3.87	15.97	12.93	60.46	82.32	141.27	179.53	158.95
init	4.18	18.17	13.85	78.53	105.63	187.90	275.03	190.17
best BB	--	19.00	13.93	91.91	119.65	222.45	340.88	195.15
final BB+	6.60	29.43	16.86	114.22	164.13	275.80	380.10	198.63
timing of the each stage								
init	0.31	2.20	0.92	18.04	23.31	46.63	95.50	31.22
best BB	--	0.83	0.08	13.38	14.02	34.55	65.85	4.98
final BB+	2.42	10.43	2.93	22.31	44.48	53.35	39.22	3.48

Table 7.3. Computing Time in Seconds

AD-A150 744 IC LAYOUT GENERATION AND COMPACTION USING MATHEMATICAL OPTIMIZATION(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE H WATANABE 1984 TR-128 N00014-78-C-0614 2/2

UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

degree of freedom. For example, a fair number of connection wires are supplied with jog points so that the compaction process can take advantage of them. The connections with a jog point consist of at least three wires rather than a single wire. This increases a number of elements in the layout and the number of 0-1 variables generated from the layout.

The eighth example, edge, is a part of an edge detector circuit for a binary image. This cell is the largest as far as the number of elements is concerned. However, the size of the graphs is not much larger than the other cells. Especially, the number of 0-1 variables is smaller than minmax and crc cells. This is because the structure of the cell is fairly simple. The horizontal metal lines constrain the height of the cell. The compaction process is therefore straightforward. The program verified the optimality of the result after the 19th branching and terminated.

In all the experiments shown in Table 7.1, the program has been terminated after the 300th branching if it could not verify the optimality of the currently known best solution. As the example of the crc indicates, the way users design an input influences the complexity of the problem; the more freedom inputs have, the more complex problems become. The number of 0-1 variables contributes to the computation time.

To find out how close the compaction result approaches to the optimal layout, the bound of the solution from the optimal should be computed. This bound can be computed from the global lower bound of the optimal solution. Since the depth first search is used, the computable global lower bound of the optimal solution is not tight. This happens because the depth first search tends to leave fairly tall subtrees unexplored. A separate program is developed for computing the bound of the compaction result from the optimal. The program computes the global lower bound of the optimal solution and explores all subtree to a given depth if necessary. This bound is computed for the examples whose compaction is terminated without verifying the optimality. The result is shown in the last row of Table 7.2.

In most cases, more than 50% of the run time was used for preprocessing, mostly for the construction of the graph structures. If cell library or predesigned cells are stored in the form of graph structures, preprocessing is not necessary. Therefore, the time needed for compaction could be reduced to less than half.

The updating of the best value is done quite frequently to a certain point. For example, in the case of the minmax cell, the best value is updated 21 times until the 84th branching, that is, an average of one update per four branching operations. Even in the most infrequent case, the case of the prcell, the best solution was updated 9 times until the 89th branching, an average of one update par ten branching operations. The current depth first branch and bound method does not find a better solution after the initial period of frequent updates. Of course, this does not mean that the method would not ultimately find a better solution, but it is no longer cost effective to continue further. This behavior of the branch and bound may be used for an automatic termination criterion. Assuming that the branch and bound process is terminated after a reasonable period of no improvement, the computation time increases linearly with the number of 0-1 variables. Therefore, the compaction method is efficient and can be used for fairly large layouts.

Table 7.4, Table 7.5 and Table 7.6 show the results of an experiment with a goal of the minimum objective dimension. Table 7.4 shows the compaction results, and Table 7.5 shows the run time information, and Table 7.6 shows the final dimensions of the four layouts. The cell used is the minmax cell, the seventh example above. The size of the problem is almost exactly the same as the one shown in Table 7.1. There is either one more x-arc or y-arc depending on the minimum objective dimension given being a width or a height. The first column shows the same data as Table 7.2 for comparison; the minmax cell is compacted as much as possible without any restrictions. The second column shows result of compaction with a fixed height of

(8) final BB -- the last branching.

(9) bound -- the worst case bound of the compaction result from the optimal.

Table 7.3 shows execution time of the above computation. It displays the following information:

(1) preproc -- computing time in cpu seconds to the end of preprocessing.

(2) init -- computing time to the end of the heuristic initial estimation process.

(3) best BB -- computing time to the end of the branching that finds the best solution.

(4) final BB+ -- computing time to the end of the branch and bound including the compacted layout output.

The input stick diagrams and the compacted layouts for all the examples are shown in Appendix B.

7.3. Discussion

The first example, scell, is a layout of a shift register cell [MeC80]. This example is the smallest. The initial estimation computed by the heuristic method is the optimal solution. The optimality is verified in the first branching and the compaction process terminates.

The second example, sbuf, is a layout for a super buffer cell [HoS80]. The third example, tff, is a layout for a t-flip-flop [Hsu79]. The number of elements in the t-flip-flop is larger than the number of elements in the super buffer cell. The super buffer cell produces more 0-1 variables than the t-flip-flop. The optimality is verified for tff but not for sbuf. Not only the size of the layout but also the structure of layout affects the size of the resulting compaction problem.

The fourth example, prcell, is a hardware implementation of a priority queue [Ked81]. This cell was originally laid out by hand. The result of compaction is 3% smaller than the one designed by hand. The fifth example, alu, is a simple bit slice ALU [DHS82]. The sixth example, minmax, is a serial comparator. It accepts as input two integers serially with the most significant bit first and outputs the larger and the smaller numbers. This cell was also originally designed by hand. The cell generated by **squash** is 2% larger than the hand designed one.

The seventh example, crc, is a cell which computes a CRC bit from a serial input. This cell is not much larger than prcell, alu, or minmax. However, its design includes a greater

	scell	sbuf	tff	prcell	alu	minmax	crc	edge
cumulative time								
preproc	3.87	15.97	12.93	60.46	82.32	141.27	179.53	158.95
init	4.18	18.17	13.85	78.53	105.63	187.90	275.03	190.17
best BB	--	19.00	13.93	91.91	119.65	222.45	340.88	195.15
final BB+	6.60	29.43	16.86	114.22	164.13	275.80	380.10	198.63
timing of the each stage								
init	0.31	2.20	0.92	18.04	23.31	46.63	95.50	31.22
best BB	--	0.83	0.08	13.38	14.02	34.55	65.85	4.98
final BB+	2.42	10.43	2.93	22.31	44.48	53.35	39.22	3.48

Table 7.3. Computing Time in Seconds

	no goal	height=100	height=115	width=120
init area	9948	9984	10925	10440
best area	7467	8500	9660	9240
improve	25%	15%	12%	12%
init iter	56	56	60	187
update	21	13	11	8
best BB	84	247	104	23
optimum	?	?	104	23
final BB	300	300	104	23

Table 7.4. Goal-directed Compaction

	no goal	height=100	height=115	width=120
cumulative time				
preproc	141.27	139.73	137.37	138.65
init	187.90	184.91	178.58	215.75
best BB	222.45	269.22	212.87	222.25
final BB+	275.80	290.78	216.13	225.43
timing of the each stage				
init	46.63	45.18	41.21	77.10
best BB	34.55	84.31	34.29	6.50
final BB+	53.35	21.56	3.26	3.18

Table 7.5. Computing Time in Seconds for Goal-directed Compaction

	Height(λ)	Width(λ)	Area(λ^2)
no goal	77	96	7392
height=100	100	85	8500
height=115	115	84	9660
width=120	77	120	9240

Table 7.6. Dimensions of ALU's

100λ. The third is with a fixed height of 115λ. The fourth is with a fixed width of 120λ. The addition of the user supplied constraints simplifies the compaction operation. At a certain point of compaction, the single simple arc from the source node to the sink node becomes the longest path of one graph. The compaction process, then, is concentrated in the other direction. In the third and fourth examples above, the program verified the optimality of the result.

CHAPTER 8

Conclusion and Future Research

8.1. Conclusion

We investigated the existing compaction algorithms, pointed out their deficiencies, and then formulated a new compaction algorithm as a mixed integer linear programming problem. A system of linear equalities and set of 0-1 variables represent the compaction problem. One efficient method for solution was presented and analyzed. This solution algorithm was based on graph algorithms. In order to solve the problem efficiently, restrictions are imposed on the formulation. The restrictions ensure that the systems of equations are encoded by almost acyclic directed graphs. Therefore, the longest path algorithm rather than the simplex method can be used for solution. The longest path algorithm, the heuristic initial estimation algorithm, and the branch and bound method provide an efficient solution method.

The new formulation of the compaction problem is more powerful than the previous models. With this formulation, the compaction problem is solved as a single two dimensional problem rather than two separate one dimensional problems. In addition, one can incorporate goals supplied by the user or a higher level composition algorithm [AcW83,TrR82]. An experimental implementation, though not coded most efficiently, produces compacted layouts within a few minutes for circuits consisting of a few hundred elements. It, therefore, demonstrates that the graph based solution algorithm is efficient enough to be useful. The compaction results are competitive with hand layout.

With the new method, library cells and leaf cells can be defined in a form of stick diagrams with parameterized transistors. One can produce customized layouts with desirable shapes, dimensions and driving capacities. The experimental program demonstrates that our compaction algorithm produces layouts of different shape from topologically similar graphs; the only difference is the size of few transistors that drive the output. They are represented by the weight of certain arcs. It also produces a layout of different shape and aspect ratio according to user supplied goals.

We can conclude that the compaction method presented is efficient and useful as a layout generation procedure for a wide range of applications. The algorithm produces sufficiently compacted layouts using a computing time roughly linear in the number of elements. The library cells for the standard cell system can be defined in graph form and can be updated according to technological changes. The library cells can also be deformed to the most preferable shape for each usage. The method can be used as a layout compaction procedure as part of an IC engineering work station [VLS82]. It can be used as a layout generation part, the final stage, of a silicon compiler [WNM83]. This is a powerful method for use in a sophisticated silicon compiler since it can change transistor sizes and the shape of each cell as necessary. For example, Trimberger developed a system for automatic timing optimization of circuit by changing transistor sizes [Tri83]. Given the optimized transistor sizes, our compaction method can produce the layout that is optimal in timing and minimal in area.

The thesis presented a graph based two dimensional compaction method. The experimental research demonstrated the practicality, generality and power of the method.

8.2. Future Research

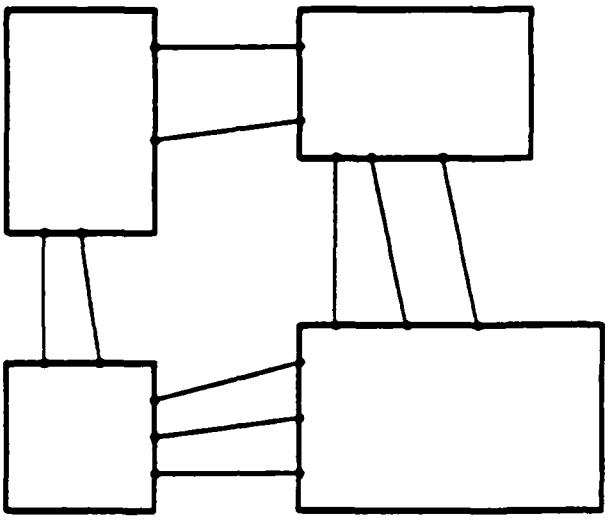
In this section, two kinds of further work are discussed. The first is work that can make the current method more complete and general. The second is work related to the integration of the compaction method into a hierarchical or more comprehensive system.

In the current program, the branch and bound method was implemented by a straightforward depth first search algorithm. It is quite possible that there is a better algorithm to implement the branch and bound method. For example, we may be better off using a hybrid of the depth first algorithm and breadth first algorithm. In order to construct the most efficient implementation for our problem, we need further experimental research.

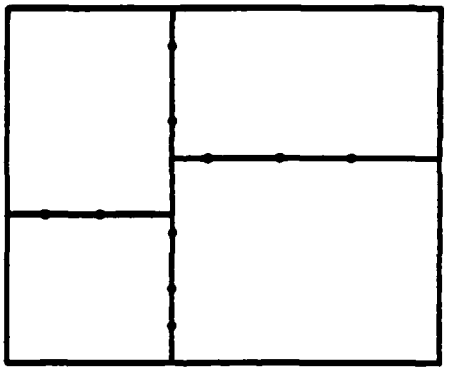
Related to the implementation of the branch and bound method is the problem of termination criteria. In the current experimental program, no sophisticated termination method was used. The user supplies the maximum number of branching operations. The easiest automatic method is to terminate the branching if the branch and bound operation continues without any improvement over a certain period. This period can be computed dynamically according to the problem complexity, the frequency of the updates, or the interval from the previous update. The depth first search method does not provide sufficient information on the difference between the current best solution and the optimal solution. The bounds we can compute are not tight enough to be useful as a termination criterion. If we have another implementation such as a hybrid of depth first search and breadth first search algorithm, we might be able to obtain much tighter bounds on the difference.

One of the main research objectives was to design a goal directed layout compaction algorithm. We successfully accomplished this objective. This enables us to direct the compaction algorithm and exercise a strong influence over the final layout. Another strong feature of our compaction method is that it enables transistors in a cell to be parameterized and the optimal layout to be produced for each case. This feature is necessary if the compacter is used as a part of silicon compilers and other sophisticated layout systems. For each application of our method, we need to develop a control and coordination algorithm for individual compaction process.

The necessity of the control algorithm is illustrated using an example of composition algorithm. A very simple minded composition program, SLAP, has been developed [Row80b]. It constructs a whole layout from leaf cells by abutting them together. In order to connect the connection points of leaf cells, it simply stretches the leaf cells. It does not take advantage of open space resulting from the stretching. Our compacter can take advantage of this space and can produce tighter layout. It is necessary to coordinate compaction of individual leaf cells, if they are to abut together in the two dimensional plane as mosaic-like construction. As a simple example, see Figure 8.1. Figure 8.1a shows four leaf cells with connection requirements among their connection points. Figure 8.1b shows the desired final layout. If we only stretch leaf cells, it is a simple task to compose these four leaf cells into the layout shown in Figure 8.1b. With our compaction algorithm, there is a possibility of producing a better layout. We need to develop an algorithm to coordinate the individual compaction to optimize the overall layout. This is only one of many possible uses of our compaction algorithm in a more general layout system. As mentioned in section 8.1, the compaction method is suitable for use in coordination with an automatic performance optimizer. Considering the flexibility and generality of our compaction method, it should be possible to find many applications for it.



a. Cells with Connection



b. Final Layout

Figure 8.1. Abuting Leaf Cells

References

[AcW83] B. Ackland and N. Weste, An Automatic Assembly Tool for Virtual Grid Symbolic Layout, in *VLSI83*, Trondheim, Norway, 1983, 457-466.

[AGR70] S. B. Akers, J. M. Geyer and D. L. Roberts, IC Mask Layout with a Single Conductor Layer, *Proceedings of 7th Design Automation Workshop*, June 1970, 7-16.

[Bak80] C. M. Baker, Artwork Analysis Tools for VLSI Circuit, Master's Thesis, Massachusetts Institute of Technology , June 1980.

[BaT80] C. Baker and C. Terman, Tools for Verifying Integrated Circuit Designs, *LAMBDA 1* ,3 (Fourth Quarter 1980), 22-30.

[Bal65] E. Balas, An Additive Algorithm for Solving Linear Programs with Zero-One Variables, *Operations Research 13* ,4 (July-August 1965), 517-546.

[BaB82] D. H. Ballard and C. M. Brown, *Computer Vision*, 1982.

[Ben62] J. F. Benders, Partitioning Procedures for Solving Mixed-Variables Programming Problems, *Numeriche Mathematik 4* (1962), 236-252.

[BHH80] J. L. Bentley, D. Haken and R. W. Hon, Statistics on VLSI Designs, CMU-SC-80-111, Department of Computer Science, Carnegie-Mellon University, April 1980.

[CiK82] M. J. Ciesielski and E. Kinnen, An Analytical Method for Compacting Routing Area in Integrated Circuits, *Proceedings of the 19th Design Automation Conference*, Las Vegas, NV., June 1982, 30-37.

[CiK83] M. J. Ciesielski and E. Kinnen, Digraph Relaxation for Placement Modification, *Proceedings of 1983 IEEE ISCAS*, Newport Beach, CA, May 1983.

[DHS82] J. DeFalcon, P. Heslin and R. Springer, THE LSI-6 A 16-Bit Minicomputer Compatible Microprocessor, *Proceedings of 1982 Custom Integrated Circuits Conference*, Rochester, N.Y., May 1982, 36-40.

[Dun78] A. E. Dunlop, SLIP: Symbolic Layout of Integrated Circuits with Compaction, *Computer Aided Design 10* ,6 (November 1978), 387-391.

[Dun79] A. E. Dunlop, *Integrated Circuit Mask Compaction*, Ph.D. Thesis, Department of Electrical Engineering, Carnegie-Mellon University, October 1979.

[Dun80] A. E. Dunlop, SLIM-The Translation of Symbolic Layouts into Mask Data, *Proceedings of 17th Design Automation Conference*, Minneapolis, Minn., June 1980, 595-602.

[GaN72] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, 1972.

[Geo69] A. M. Geoffrion, Integer Programming by Implicit Enumeration and Balas' Method, *SIAM Review 7* ,2 (April 1969), 178-190.

[GeG71] A. M. Geoffrion and G. W. Graves, Multicommodity Distribution System Design by Benders Decomposition, *Management Science 20* ,5 (January 1971), 822-844.

[GiN76] D. Gibson and S. Nance, SLIC - Symbolic Layout of Integrated Circuits, *Proceedings of 13th Design Automation Conference*, June 1976, 434-440.

[GiN77] D. Gibson and S. Nance, Symbolic System for Layout and Checking, *Proceedings of 1977 IEEE ISCAS*, Phoenix, 1977, 436-440.

[Hac81] G. D. Hachtel, On the Sparse Tableau Approach to Optimal Layout, *Proceedings of 1981 IEEE ISCAS*, 1981, 1019-1022.

[Hel79] W. R. Heller, An Algorithm for Chip Planning, SSP Memo #2806, Computer Science Department, California Institute of Technology, May 1979.

[HoS80] R. W. Hon and C. H. Sequin, A Guide to LSI Implementation, Second Edition, SSL-79-7, Palo Alto Research Center, XEROX, January 1980.

[HsP79] M. Hsueh and D. O. Pederson, Computer-Aided Layout of LSI Circuit Building-Blocks, *Proceedings of 1979 International Symposium on Circuits and Systems*, Tokyo, 1979, 747-477.

[Hsu79] M. Hsueh, Symbolic Layout and Compaction of Integrated Circuit, Memorandum No. UCB/ERL M79/80, Electronics Research Lab., University of California, Berkeley, December 1979.

[Ked81] G. Kedem, A First In, First Out and a Priority Queue, Tech. Rep.-90, Computer Science Department, University of Rochester, March 1981.

[LaP73] A. H. Land and S. Powell, *Fortran Codes for Mathematical Programming: Linear, Quadratic and Discrete*, 1973.

[Lau79] U. Lauther, A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation, *Proceedings of 16th Design Automation Conference*, June 1979, 1-10.

[Law76] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, 1976.

[LRS83] C. E. Leiserson, F. M. Rose and J. B. Saxe, Optimizing Synchronous Circuitry by Retiming, *Third Caltech Conference on Very Large Scale Integration*, 1983, 87-116.

[LiW83] Y. Liao and C. K. Wong, An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-2* ,2 (April, 1983), 62-69.

[Lyo80] R. F. Lyon, Simplified Design Rules for VLSI Layouts, *LAMDA, The Magazine of VLSI Design 2* ,1 (First Quarter, 1980), 54-59.

[McW78a] T. M. McWilliams and L. C. Widdoes, SCALD: Structured Computer-Aided Logic Design , *Proceedings of 15th Design Automation Conference*, June 1978, 271-277.

[McW78b] T. M. McWilliams and L. C. Widdoes, The SCALD Physical Design Subsystem, *Proceedings of 15th Design Automation Conference*, Las Vegas, Nevada, June 1978, 278-284.

[MeC80] C. Mead and L. Conway, *Introduction to VLSI Systems*, 1980.

[Mos80] R. C. Mosteller, REST User Guide, SSP File #4030, Computer Science Department, California Institute of Technology, October 1980.

[Mos81] R. C. Mosteller, REST A Leaf Cell Design System, in *VLSI81 Very Large Scale Integeration*, J. P. Gray (ed.), Academic Press , 1981, 163-172.

[OSK70] T. Ohtsuki, N. Sugiyama and H. Kawanishi, An Optimization Technique For Integrated Circuit Layout Design, *Kyoto International Conference on Circuit and System Theory, Proceedings*, Kyoto, 1970, 67-68.

[Otv75] R. H. J. M. Otten and M. C. van Lier, Automatic IC-Layout: The Geometry of The Islands, *Proceedings 1975 IEEE International Symposium on Circuits and Systems*, Newton, Mass., April 1975, 231-234.

[PDS77] D. Persky, D. N. Deutsch and D. G. Schweikert, LTX -- A Minicomputer-Based System For Automated LSI Layout, *Journal of Design Automation and Fault-Tolerant Computing 1* ,3 (May 1977), 217-255.

[Prv79] B. T. Preas and W. M. vanCleemput, Routing Algorithms for Hierarchical IC Layout, *Proceedings of 1979 International Conference on Circuits and Systems*, Tokyo, 1979, 482-485.

[Pre79] B. T. Preas, *Methods for Hierarchical Automatic Layout of Custom LSI Circuit Masks*, Ph.D. Thesis, Department of Electrical Engineering, Stanford University, August 1979.

[PrG79] B. T. Preas and C. W. Gwyn, General Hierarchical Automatic Layout of Custom VLSI Circuit Masks, *Journal of Design Automation and Fault-Tolerant Computing 3* ,1 (January 1979), 41-58.

[Row80a] J. Rowson, Geometry Composition -- Another Look , SSP File #3792, Computer Science Department, California Institute of Technology, June 1980.

[Row80b] J. Rowson, *Understanding Hierarchical Design*, Ph.D. Thesis, Computer Scienc Department, California Institute of Technology, April 1980.

[SLM82] M. Schlag, F. Luccio, P. Maestrini, D. T. Lee and C. K. Wong, A Visibility Problem in VLSI Layout Compaction, RC 9896, IBM T. J. Watson Research Center, 1982.

[SzO80] A. A. Szepieniee and H. J. M. Otten, The Genealogical Approach to the Layout Problem, *Proceedings of 17th Design Automation Conference*, Minneapolis, Minn., June 1980, 535-542.

[Ter80] C. Terman, C Program for Node Extraction , Distributed by MIT, 1980.

[TrR82] S. Trimberger and J. Rowson, Riot -- A Simple Graphical Chip Assembly Tools, *Proceedings of the 19th Design Automation Conference*, Las Vegas, NV., June 1982, 371-376.

[Tri83] S. Trimberger, Automated Performance Optimization of Custom Integrated Circuits, *Proceedings of 1983 IEEE ISCAS*, Newport Beach, CA., May 1983, 194-197.

[VLS82] VLSI Technology Inc., *The Integrated VLSI Design System*, VLSI Technology Inc., 1982.

[Wes81a] N. Weste, Virtual Grid Symbolic Layout, *Proceedings of 18th Design Automation Conference*, June 1981, 225-233.

[Wes81b] N. Weste, MULGA -- An Interactive Symbolic Layout System for the Design of Integrated Circuits, *The Bell System Technical Journal 60* ,6 (July-August 1981), 823-857.

[Wil77] J. D. Williams, *STICKS--A New Approach To LSI Design*, Master's Thesis, Massachusetts Institute of Technology, June 1977.

[Wil78] J. D. Williams, STICKS--A Graphical Compiler for High Level LSI Design, *AFIPS Conference Proceedings 47* (June 1978), 289-295.

[WNM83] W. Wolf, J. Newkirk, R. Mathews and R. Dutton, Dumbo : A Schematic-to-Layout Compiler, *Third Caltech Conference on Very Large Scale Integration*, 1983, 379-393.

[ZiS74] K. Zibert and R. Saal, On Computer Aided Hybrid Circuit Layout, *Proceedings 1974 IEEE International Symposium on Circuits and Systems*, San Francisco, CA., April 1974, 314-318.

APPENDIX A

Documentation for Squash User

In the following, all information written in italics is only for a University of Rochester user.

The compaction program **squash** is developed for experimental purposes as a research tool. It is not user friendly. To be honest, it is outright hostile. The input of the program is loosely drawn layout in CIF. This CIF input must agree with a defined format, any deviation from the required format causes **squash** to terminate. The output is also in the CIF format.

Following are the brief guidelines to prepare an input for **squash.**

Input consists of symbols and wires. Symbols are system-defined library symbols. Wires are diffusion, polysilicon and metal line segments, which connect symbols. The user is not allowed to create any active element by himself.

Pullup transistors (number indicate length/width ratio):
pup1, pup2, pup3, pup4, pup5, pup6, pup7, pup8, pup9, pup10

Enhancement mode transistors:
etrn1, etrn1/2, etrn1/3, etrn1/4, etrn1/5, etrn1/6,
etrn1/7, etrn1/8, etrn1/9, etrn1/10

Depletion mode transistors:
dtrn1, dtrn2, dtrn3, dtrn4, dtrn5, dtrn6, dtrn1/2

Buried contacts:
bcont1 (poly and diffusion crossing)
bcont2 (poly surround diffusion)
bcont3 (diffusion surround poly)

Poly-metal contact:
pcont
pcont2 (size is twice larger than pcont)

Diffusion-metal contact:
dcont
dcont2 (size is twice larger than dcont)

Pullup Transistor with butting contacts:
btpup2, btpup3

Butting contact:
btcont

If you would like to save symbol space of ICARUS, you can delete any unnecessary symbols from the library. Deleted symbols reappear after processing by squash. Therefore, if you are low in symbol space use the "dic" program to extract only necessary symbols, or delete unnecessary symbols in ICARUS.

Rules concerning wires.

(1) Diagonal wires are not allowed.

(2) A wire must be either horizontal or vertical. **Squash** extracts this information from the shape of the rectangle. The user should not draw a square box for a wire segment in ICARUS. A horizontal wire must be a rectangle longer in the horizontal direction.

(3) A wire must terminate by contacting other elements. Other elements are either symbols, other wire segments in orthogonal direction, or boundaries. Wires in the same direction are not allowed to connect to each other. If they do, that connection is ignored. Boundaries are created by the program; they are edges of the smallest box that surrounds everything. Make sure that all of your incoming and outgoing wires (i.e. input and output to outside, clock lines, ground and vdd lines) reach the boundaries. *You can use the symbol definition facility in ICARUS to make sure that you do not have a wire which does not reach the boundary of your layout.*

(4) The user can cross a metal wire and a poly wire, and a metal wire and a diffusion wire. Otherwise, do not cross any two wires.

(5) You can draw wires with less than legal width, for example 1 lambda. Wires are expanded to the minimum width required. If you want a 4 lambda metal wire, draw as a metal wire with 4 lambda width.

(6) Do not terminate more than 2 wires in the same place. That is, you should not make T crossings of wires with three wires. The user can have a T crossing consisting of two wires

Rules concerning symbols

(1) Do not connect two symbols directly. Two symbols must be connected by a wire.

(2) Do not connect more than one wire in one side of a symbol. You can connect a maximum of four wires per single symbol. If you need more than one wire in the same side and if they are in the same layer, use a wire segment to branch out more than one wire. See Figure A.1. If they are in different layer, go around from other sides. See Figure A.2.

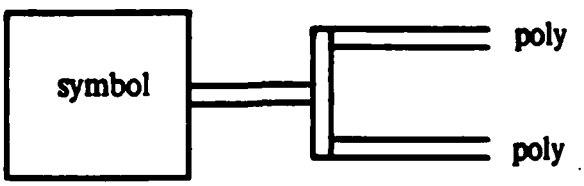


Figure A.1. Two Wires in Same Layer

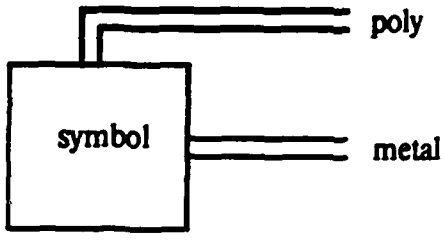


Figure A.2. Two Wires in Different Layers

Rules concerning symbols defined by a user

Connection to the outside of a user defined symbol causes some problem, since wires in the same direction are not allowed to connect to each other. A vertical wire must be connected to a horizontal wire or vice versa. If you want to connect wires in the same direction, you can creat a short wire in the orthogonal direction and overlap it over a connection point of the original two wires. Its length is equal to the width of the other two wires, and its width is half of the regular width. Since original wires touch each other, **squash** will produce an error message, but if everything else is correct this does not affect the program. It gives more freedom to the compacter. It introduces a jog point in a straight wire and the compacter can introduce a bend if it is advantageous.

Other rules

The method described to connect two wires in the same direction is also useful as a method to introduce a jog point in a straight wire. You can introduce a bend in a certain wire in the layout after you have inspected a compacted layout. Just replace a wire with two wire segments and overlap a short wire in the orthogonal direction over the contact point.

How to run program

Use the design rule checker, before you run the compacter. In order to run program, type:

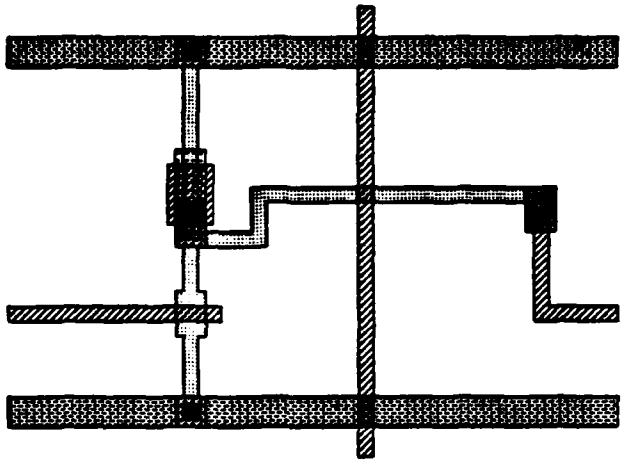
squash file num > junk

file is your cif file (name.cif) typed without the extension (.cif)
num is an integer number representing the maximum number of branching operations. Try 200-1000.
junk is a listing file. **Squash** is still changing, and for debugging and diagnostic purposes, it prints out a lot of junk.

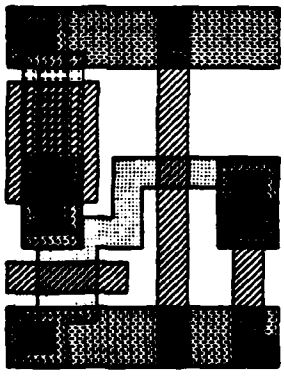
The result of compaction is always in a file "compact1.cif". If mgen dies, search "***" in your listing file. You will find an error statement.

APPENDIX B

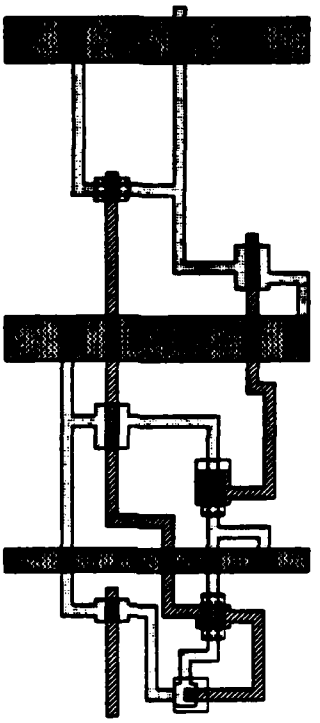
Compaction Examples



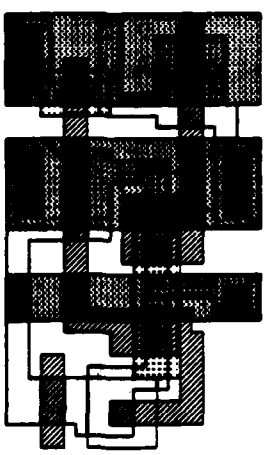
scell input



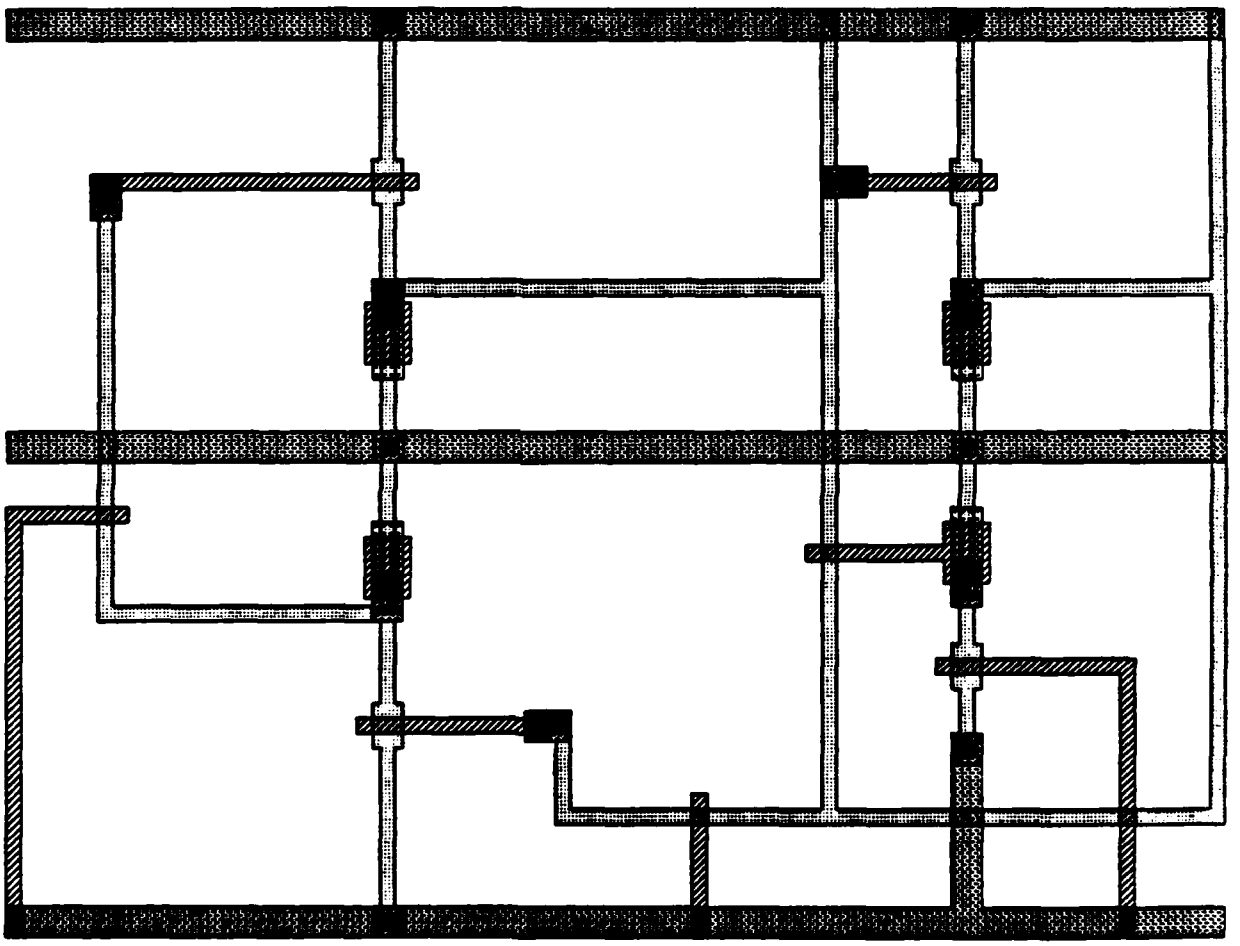
scell output



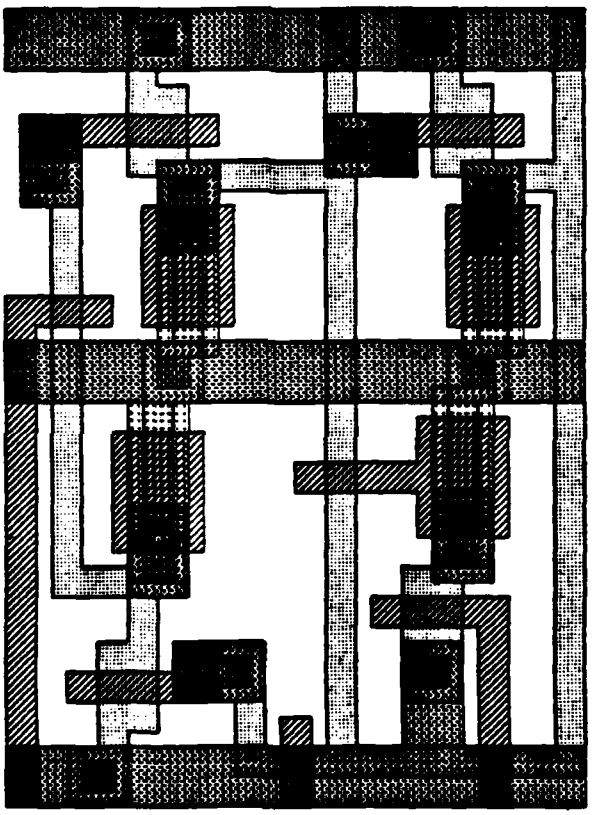
sbuf input



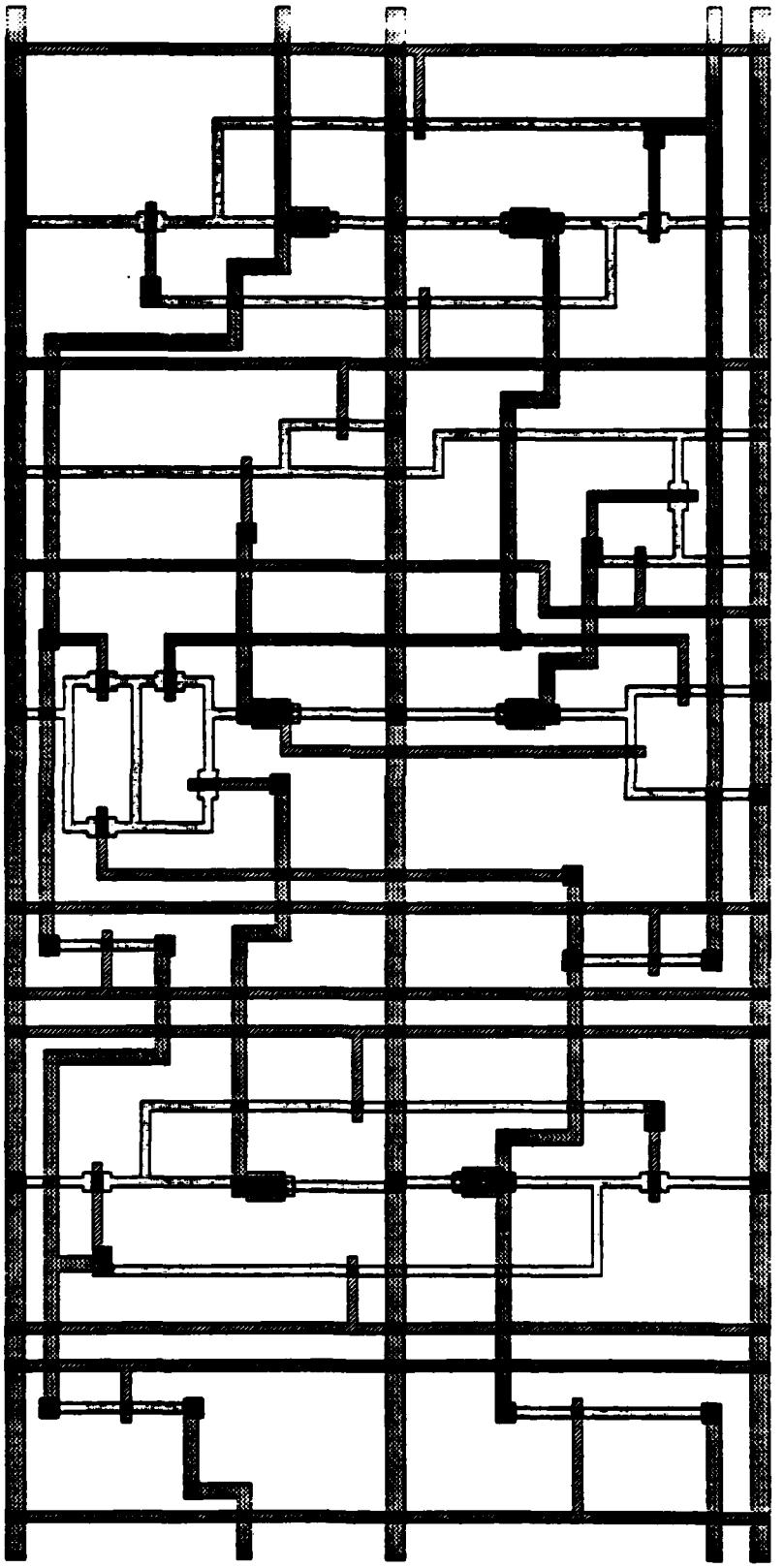
sbuf output



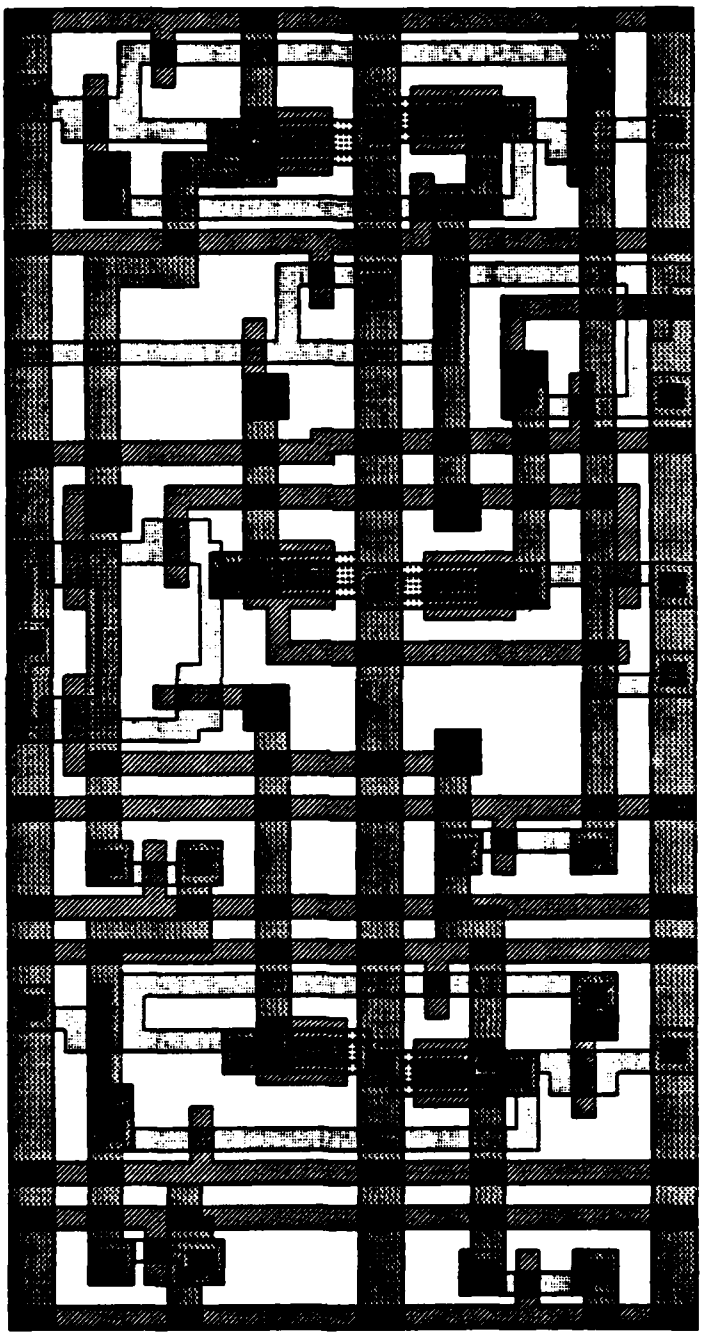
tff input



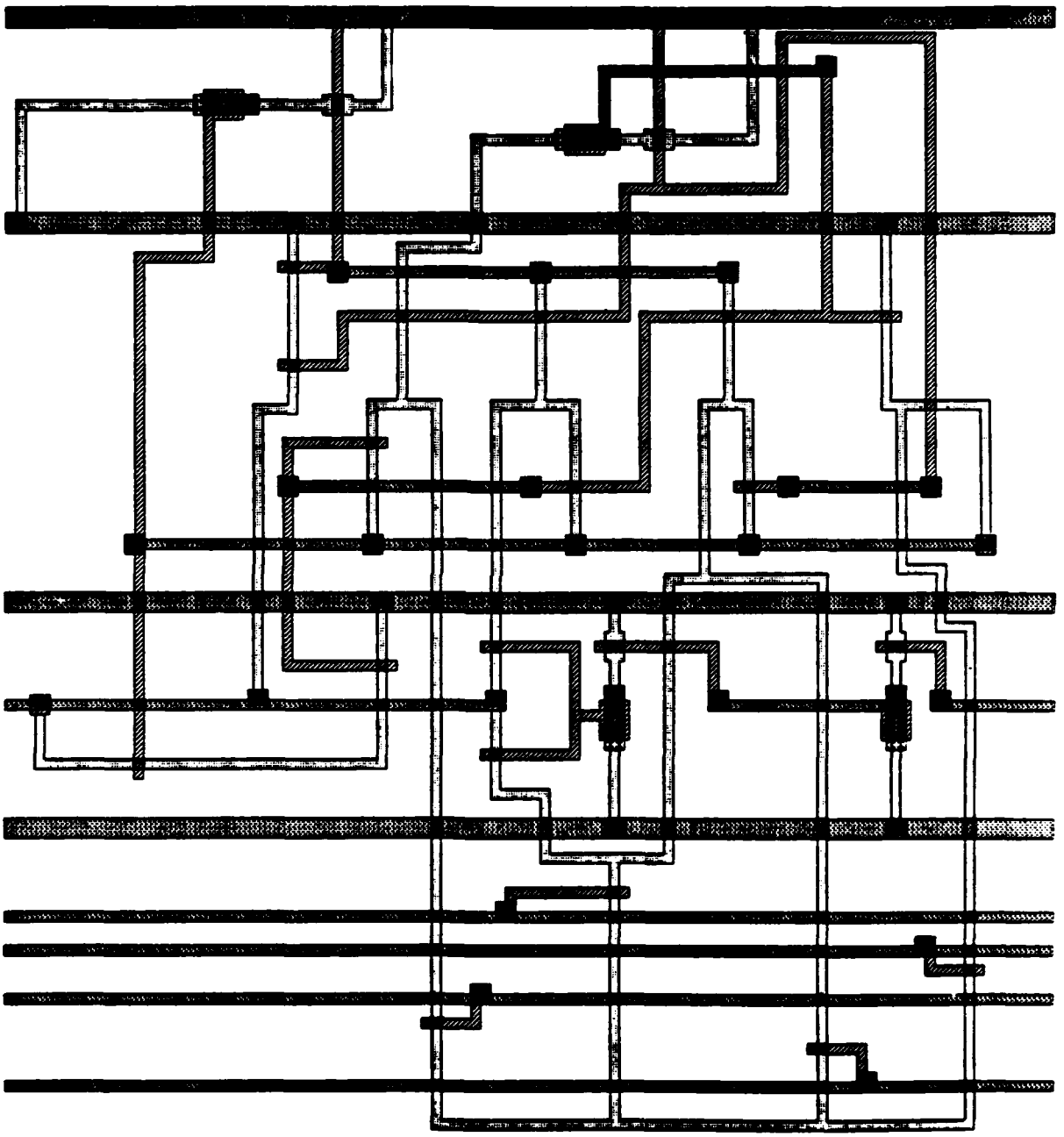
tff output



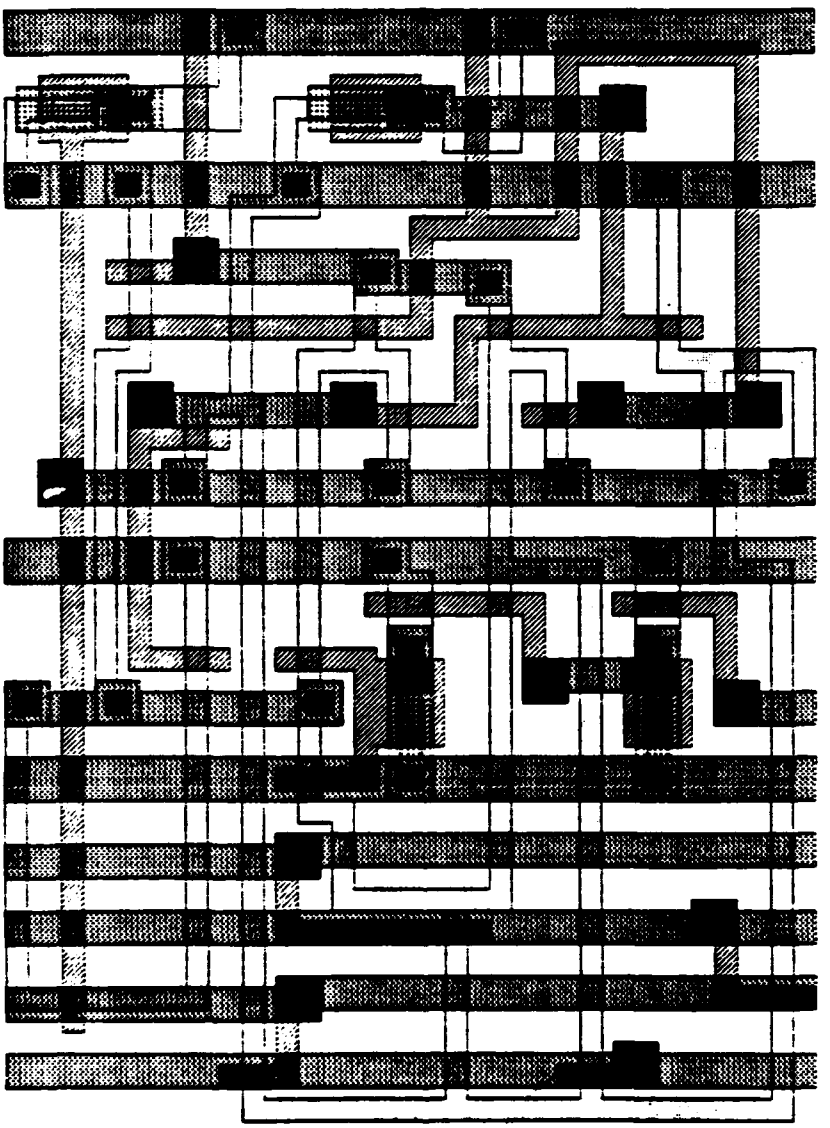
prcell input



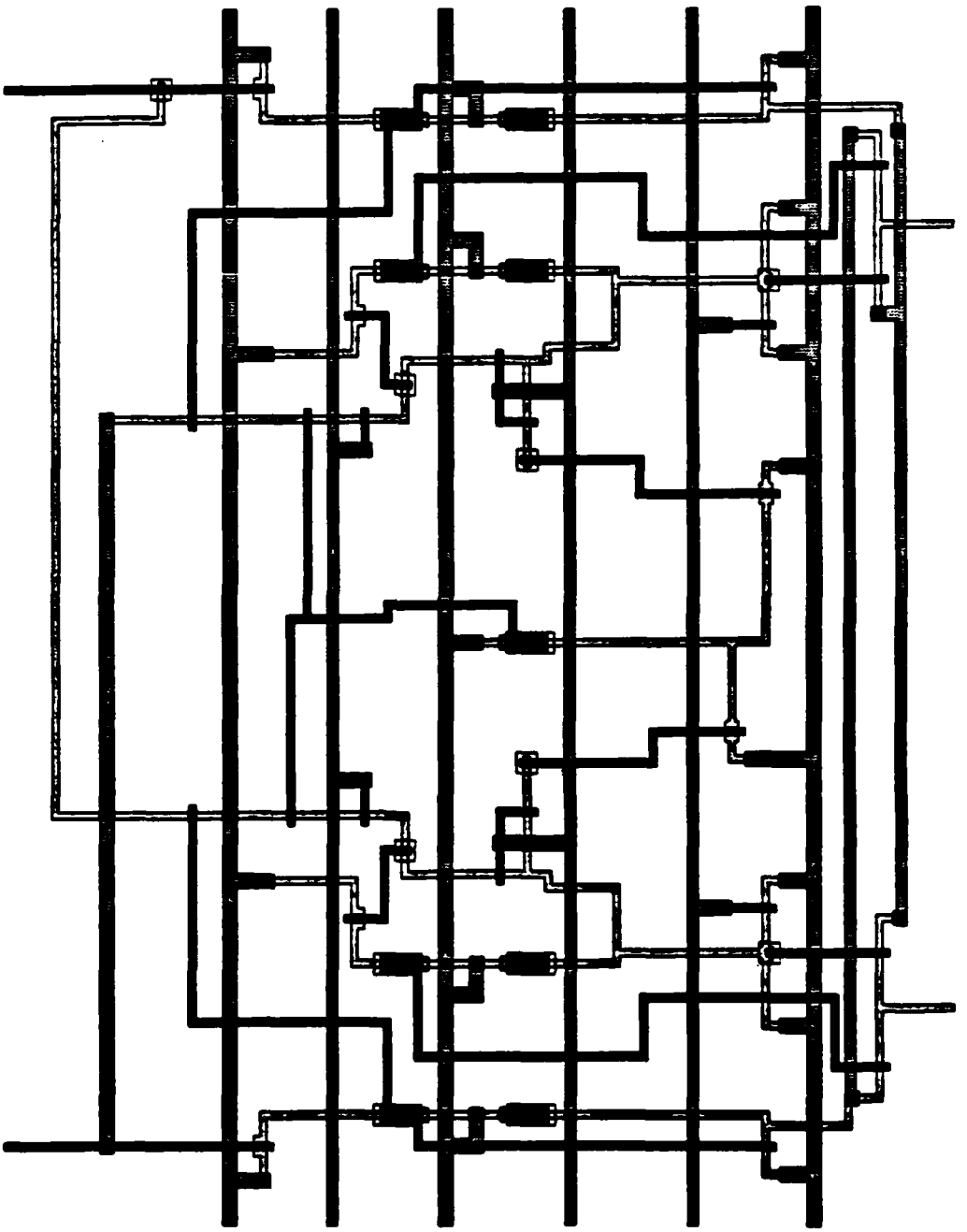
prcell output



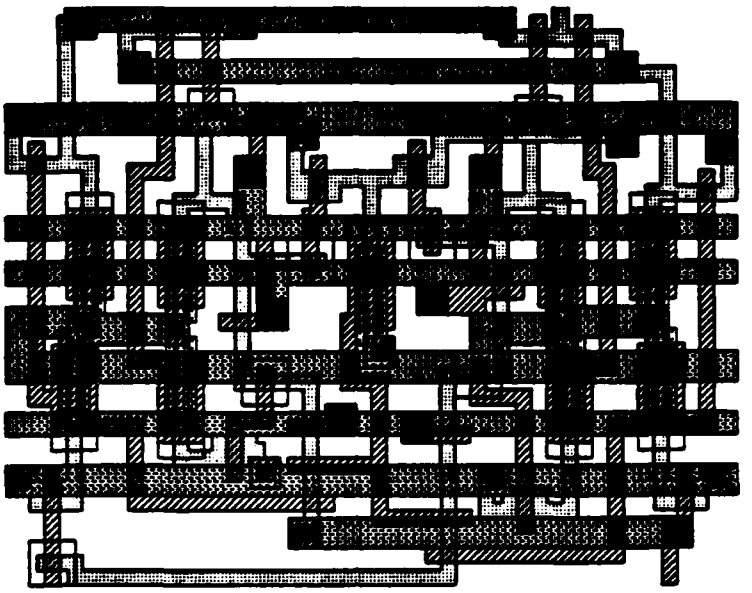
alu input



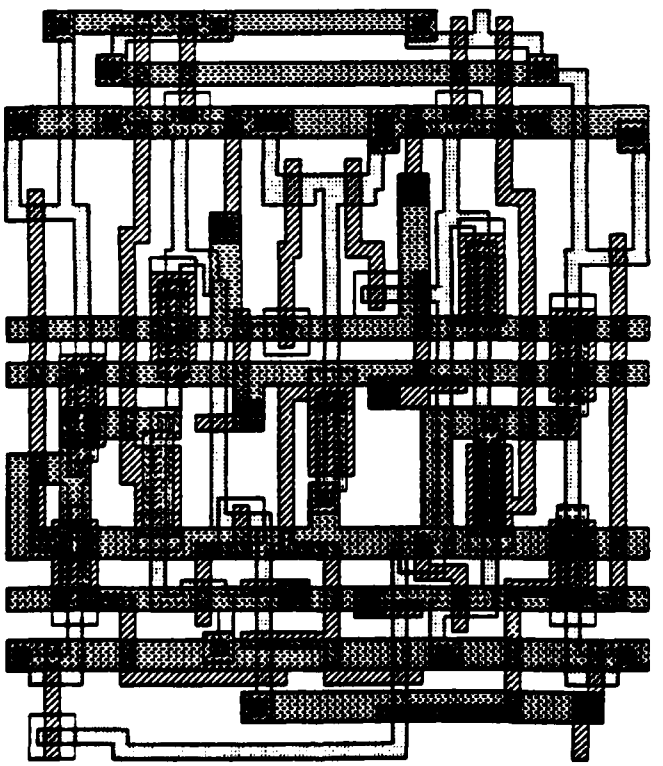
alu output



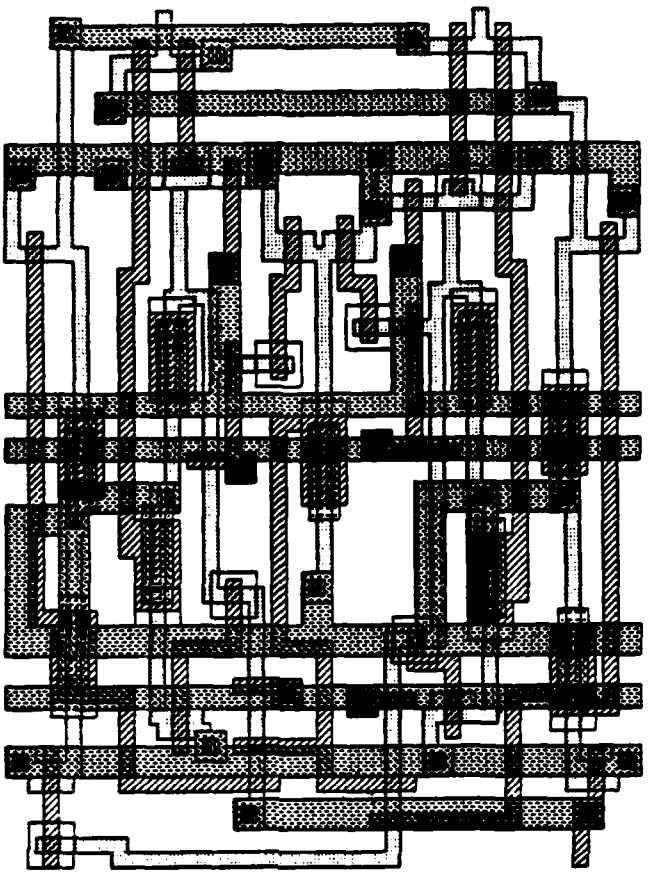
minmax input



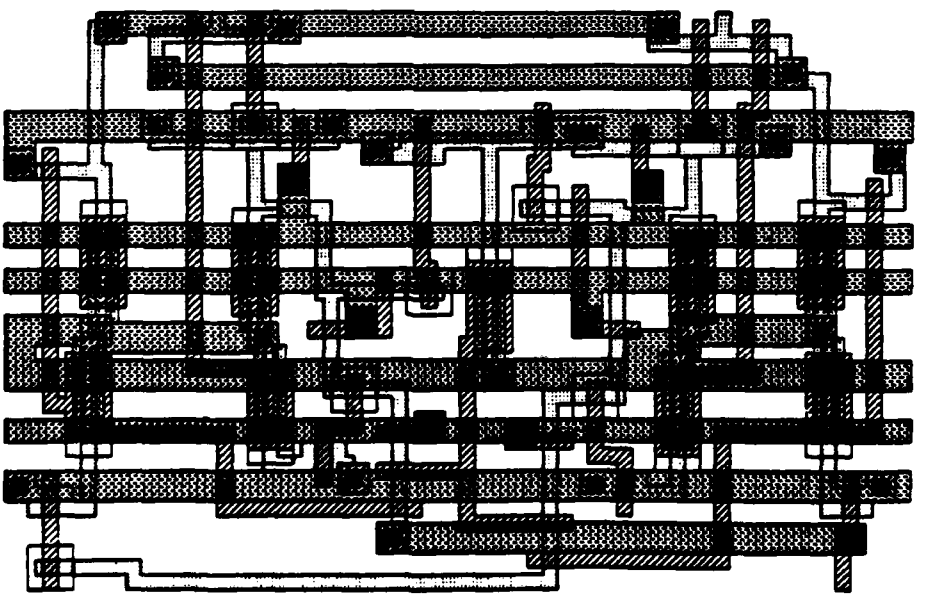
minmax ouput



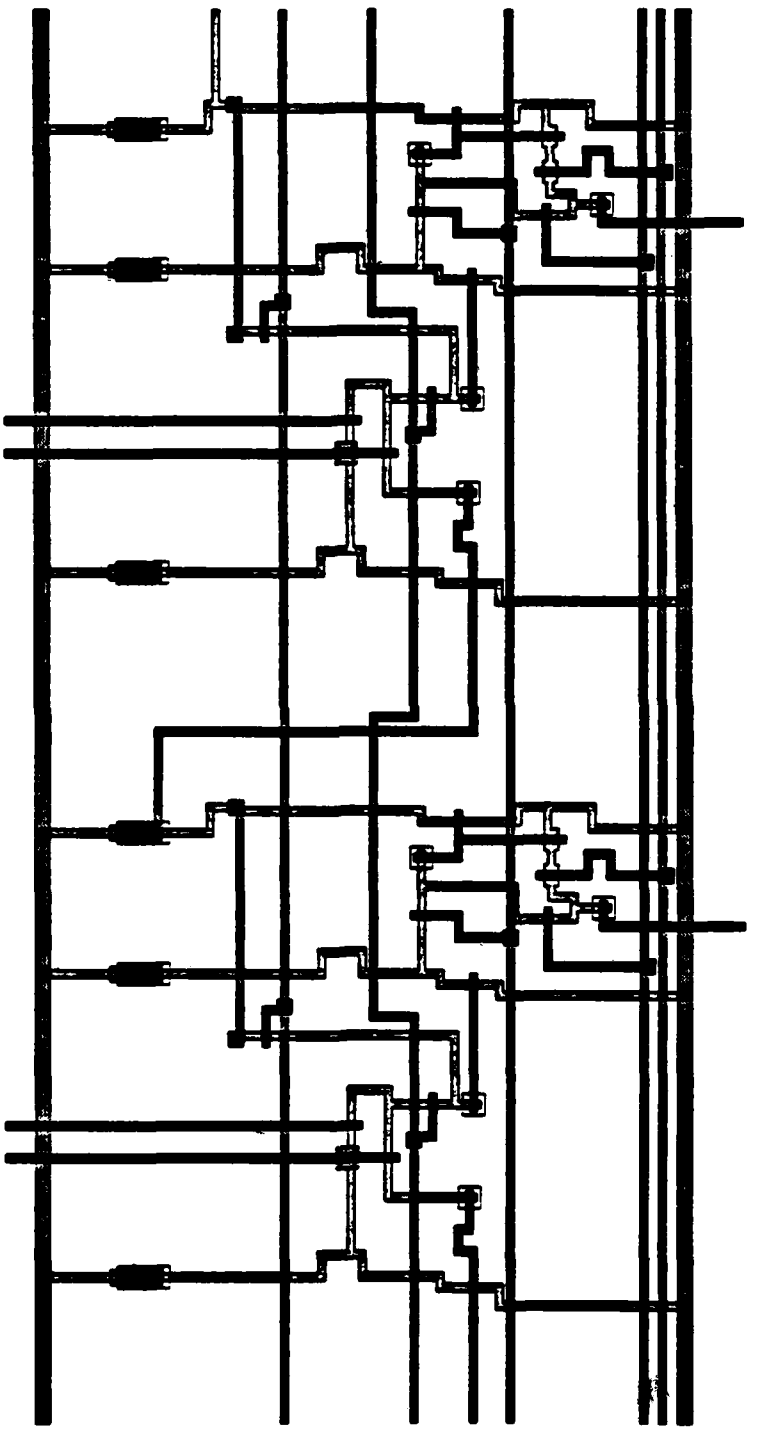
minmax output with height=100



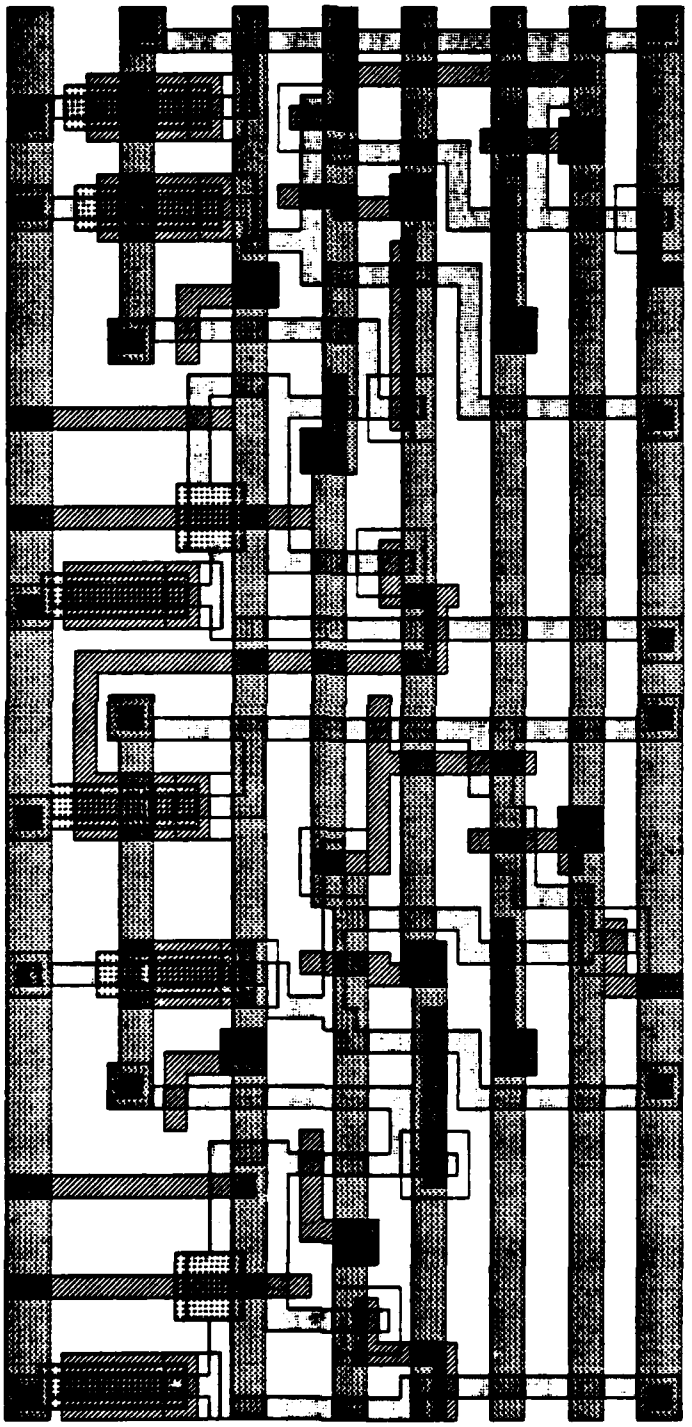
minmax output with height=115



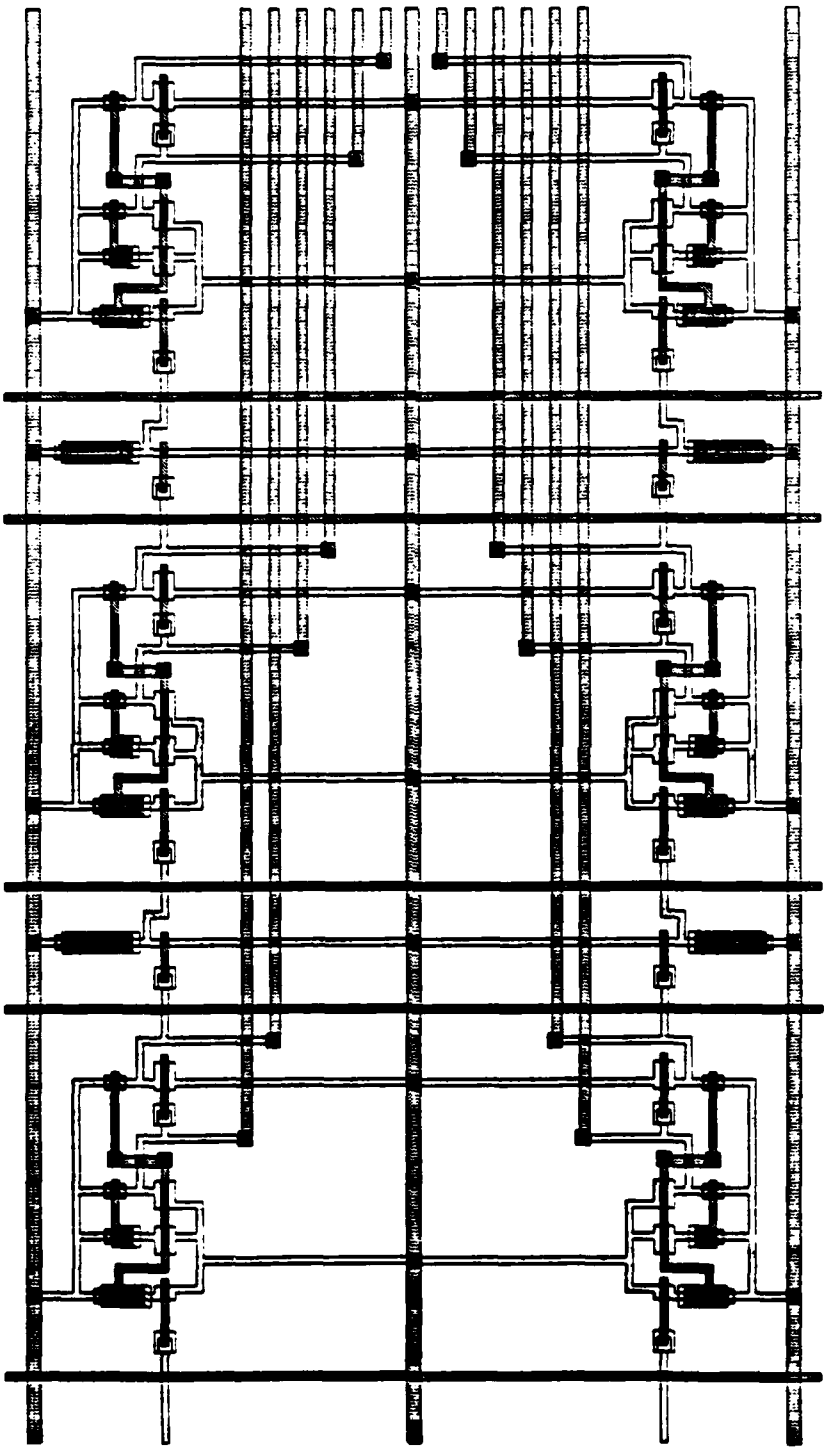
minmax output with width=120



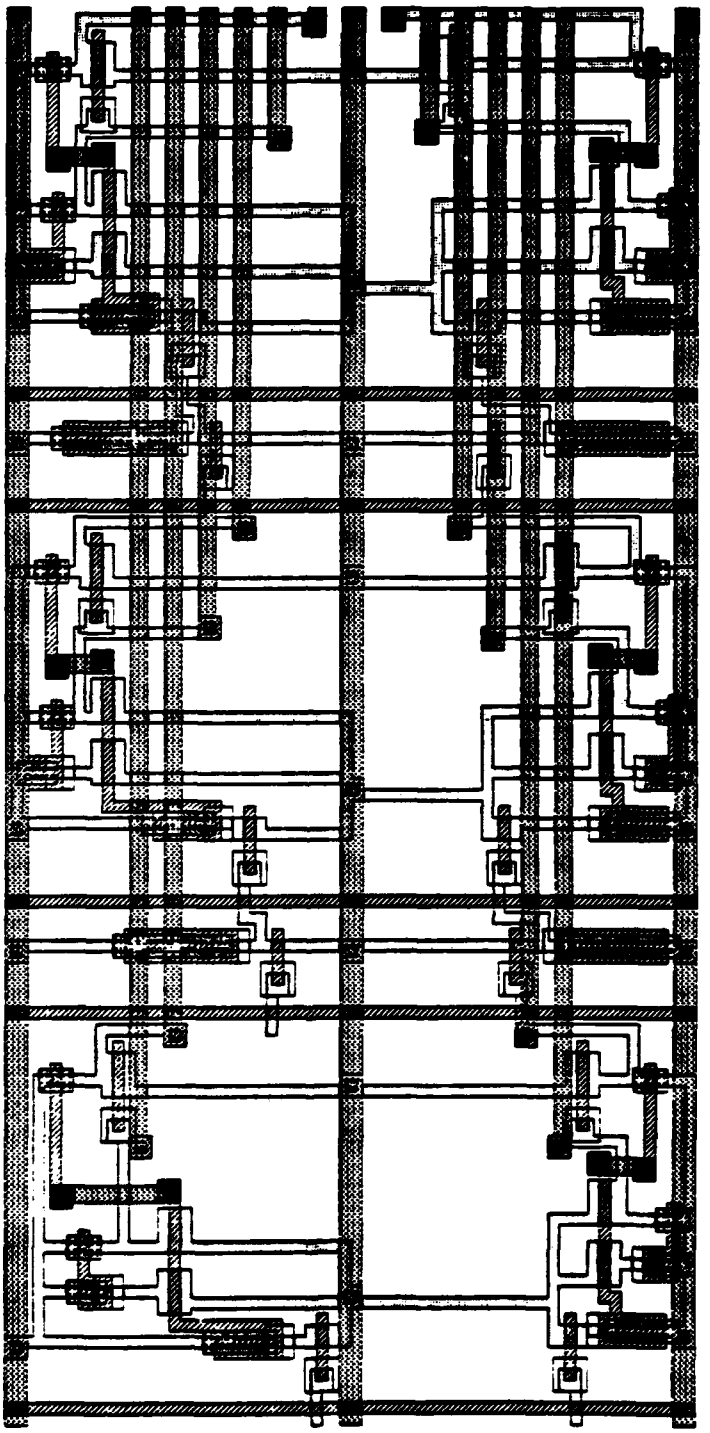
crc input



crc output



edge input



edge output

END

FILMED

3-85

DTIC